F. DACOSTA

A KALANDPROGRAM ÍRÁSÁNAK REJTELMEI

HOGYAN ÍRJUNK BASIC NYELVEN AZ ISKOLA-SZÁMÍTÓGÉPRE KALANDPROGRAMOT

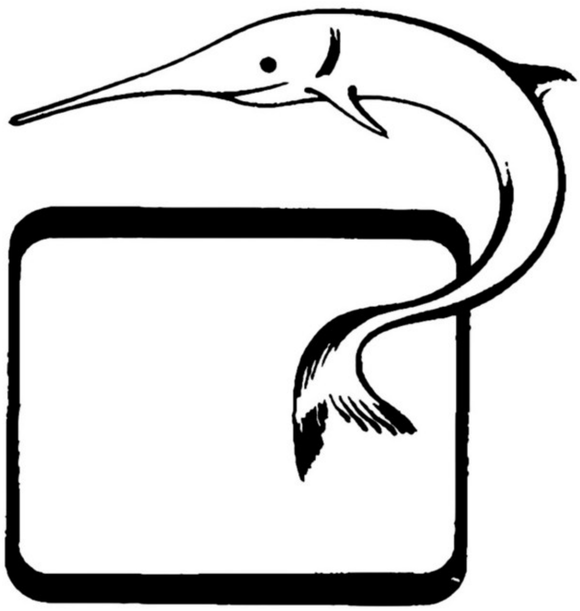


MŰSZAKI KÖNYVKIADÓ

F. DACOSTA

A KALANDPROGRAM ÍRÁSÁNAK REJTELMEI

HOGYAN ÍRJUNK BASIC NYELVEN AZ ISKOLA-SZÁMÍTÓGÉPRE KALANDPROGRAMOT



MŰSZAKI KÖNYVKIADÓ, BUDAPEST, 1986

Az eredeti mű:
F. DaCosta: Writing BASIC Adventure programs for the TRS–80

Original U.S. edition published by TAB BOOKS Inc.
Blue Ridge Summit, Penna.

Fordította: BÁN PÉTER
Lektorálta: DR. FUTÓ ISTVÁN

ETO: 681.3.06
800.92. BASIC

ISBN: 963 10 6704 1

Felelős szerkesztő: VOTISKY ZSUZSA

Tartalom

Előszó a magyar kiadáshoz

A könyv eredetije angol nyelven íródott. A játékok jellege folytán nem lehetett mindig szó szerinti fordítást készíteni. Egészen pontosan: az első játéknál — az eredetiben — az angol nyelvtannak többé-kevésbé megfelelő formában kellett a parancsokat leírni, a lefordított formában pedig természetesen magyarul. A „többé-kevésbé" kifejezés arra utal, hogy egy meglehetősen egyszerű játékprogram nem tűzhette ki céljául, hogy bonyolult nyelvi fordulatokat elemezzen, felismerjen és helyesen végrehajtson. Az eredeti játék parancsai ilyenek voltak:

WAIT (várj) vagy EAST (kelet), ill.
DROP AXE (to drop = elejteni, axe = szekerce).

Az egyszavas parancsokat természetesen magyarul is változtatás nélkül használhatjuk. A nehézséget a többszavas parancsok okozzák. Míg az angol nyelvben talán elfogadható a többszavas parancsok formája, ez a magyarban már semmiképpen sem „megy el". Pl. hogyan fordíthatnánk a fenti példát:

FELVENNI SZEKERCE vagy SZEKERCÉT FELVENNI?

Az egyik értelmetlen, a másik suta.
Jobban illeszkedik a magyar nyelvtan szabályaihoz, ha nem az első és a második, hanem az első és az utolsó szót figyeljük. A példában ez így hangzik:

VEDD FEL A SZEKERCÉT

A parancsok nyelvtana tehát úgy változott, hogy többszavas parancsoknál a második szó helyett az utolsó szót tekinti lényegesnek a program. Ebből természetesen az is következik, hogy az első és az utolsó szó közé írt szöveget a program figyelmen kívül hagyja: ha valaki siet a játékban, bátran leírhatja:

VEDD A SZEKERCÉT vagy
VEDD SZEKERCÉT

A második változtatást az tette szükségessé, hogy az angol nyelvben elöljárókat használnak, a magyar nyelvben pedig magát a szót megváltoztató ragokat. A szótárban tehát az öt betűnél rövidebb szavakat ragjukkal együtt kellett fölvenni. Pl.: DÉL, MENJ DÉLRE vagy MENJ DÉLNEK. Hasonlóan PÓKOT, PÓKKAL, mivel a játékos írhatja azt is, hogy ÖLD MEG A PÓKOT vagy KÜZDJ A PÓKKAL.

A második játékban a magyarítás ilyen nehézséget nem okozott, mivel ott egybetűs parancsokkal vezéreljük a játékot. Itt a játék lassúsága miatt kellett változtatni. Az iskola-számítógép nem tud elég gyorsan reagálni a leütött billentyűkre. Pontosabban: ebben a játékban igen gyakran kell a nyíllal jelölt billentyűket használni. Ez csak abban az esetben kényelmes, ha a billentyűket nem kell szaporán ütögetni, elég egyszer lenyomni, majd lenyomva tartani. Ilyenkor viszont a gép nagyon lassan reagál. Ez tette szükségessé a változtatást. Végül, kihasználva az iskola-számítógép nyújtotta lehetőségeket, az üzenetek és a parancsok az ékzetes betűket használják. Így a SZÖRNYEK AZ ÚTVESZTŐBEN c. játékban a játékost „ə" jel helyett egy „É" betű jelzi.

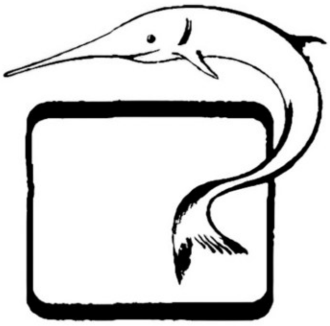
Bevezető

Aki szereti a kalandot, és a legbonyolultabb útvesztőből is kitalál, aki még értéktelen kincsek elnyeréséért is képes „borzalmas" veszélyeket leküzdeni, az vérbeli számítógépes kalandozó! Az ilyen emberek bizonyára gondoltak már arra, hogy ők maguk készítsenek kalandprogramot.

Nos, e könyv segítségével valódi kalandjátékot készíthetünk BASIC nyelven. Mindezekhez egy 16 kbyte-os TRS–80 Model I vagy III mikroszámítógép* és egy mágnesszalagos háttértár kell. A szükséges tanácsok és példák a könyvben megtalálhatók. Két példaprogramot mutatunk be: a *Kardhalak és kincsek*-et és a *Szörnyek az útvesztőben*-t. Felépítésüket a legapróbb részletekig elmagyarázzuk, a programvégrehajtási ciklustól kezdve a szubrutinokon, az egyes vezérlőrészeken át a parancsokig. Megmutatjuk, hogyan használhatunk gépi kódú szubrutinokat, hogy felgyorsítsuk a BASIC-ben írt programot. Megismerkedhetünk az indexes változók alkalmazásának és az információtárolásnak több módszerével. Nagyobb nehézségek nélkül elsajátíthatjuk a strukturált programozás logikus gondolkodásmódját.

Egyszóval, könyvünk áttanulmányozása serkenti az Olvasó képzeletét, segíti abban, hogy önmaga is készíthessen kalandprogramot.

*Magyarországon ennek a HT-1080Z felel meg. (*A fordító*)



1. FEJEZET

Kalandozás a billentyűk mellett

A táblás játékok gyártói néhány éve meglepő felfedezést tettek. Rájöttek, hogy nagy piaca van a bonyolult, szerepjátszó játékoknak. A játékosok már nem elégedtek meg azokkal a játékokkal, amelyekben egy bábuval lépkedtek a táblán, olyan játékokra vágytak, amelyekben érvényesíthették képzelőerejüket, stratégiai érzéküket. Ennek nyomán született meg a táblás játékok új nemzedéke, amelyekben a játékos történelmi csatákat játszhatott le, megküzdhetett sárkányokkal vagy képzelet szülte seregekkel.

Később beköszöntött a „házi" számítógépek kora, és a fantáziajátékok hívei azt jósolták, hogy nincs már messze az az idő, amikor a kettő szövetségre lép. Hiszen végeredményben a fantázia és a szimulációs játékok igen bonyolultak, és előfordul, hogy a bonyolult játékszabályok betartása a játék színvonalának rovására megy. Ha azonban egy mikroszámítógép tartaná számon a találatokat, mozgatná a bábukat, írná le az állást, nos, akkor a játékos valóban a játékkal törődhetne. Ideális szövetség!

A szövetség tényleg létrejött, de a végleges irányt még valami befolyásolta; a nagyszámítógépeken futó fantázia- és játékprogramok. Már azóta is jó néhány év eltelt, hogy Crowther és Woods bemutatta Adventure (Kaland) nevű programját. Ebben a szórakoztató szimulációs játékban a játékos egy veszélyekkel teli barlangba kerül, manókkal és kígyókkal harcol, miközben kincset keres. A kalandjátékoknak ezt a prototípusát PDP–10 számítógépre írták. Az egyetemisták kedvenc időtöltése volt, jóval azelőtt, hogy mikroszámítógépes változata megjelent volna.

Manapság mindegyik amatőr számítógépes folyóiratban sok játékprogramot találunk, kalandprogramot és szimulációs programot egyaránt. Némelyikük hasonlít az eredeti Adventure-re, másokban új fogások vannak. Némelyik BASIC-ben íródott, hiszen a legtöbb személyi számítógépben ROM-ba égetett BASIC van, mások gyors és hatékony gépi kódú programok.

Feltűnt, hogy senki sem veszi a fáradságot, hogy megmagyarázza: hogyan

is működnek ezek a kalandprogramok — vagyis a dolog programozási oldalát —, és hogyan is írhatnánk magunk is ilyen programokat.

Nos, e könyv éppen ezzel foglalkozik! A könnyebb érthetőség kedvéért a közölt kalandprogram BASIC nyelven íródott, nem gépi kódban. Ez fontos, ugyanis a legkülönbözőbb ötletes fogásokra lesz szükség annak érdekében, hogy a testes BASIC program elég hatékonyan működjön.

A könyvben a „kalandprogram" kifejezést használjuk minden olyan programra, amelyben a játék általános szerkezete megegyezik Crowther és Woods eredeti programjáéval. A magyarázatok és a példák nem valamelyik konkrét kereskedelmi forgalomban kapható program kódjából származnak, hanem egy teljesen új, kifejezetten e könyv céljaira készült programot közlünk, *Kardhalak és kincsek* címmel. Célunk az volt, hogy programozási ismereteket tanítsunk, nem pedig az, hogy lehetetlenné tegyük a többi, keményen dolgozó programozó számára programjaik értékesítését. Ráadásként egy másik kalandprogram is van a könyv 11–13. fejezetében, a *Szörnyek az útvesztőben* című. Ez másfajta program, kihasználja a TRS–80 grafikájában rejlő lehetőségeket, így real-time kalandokkal is kísérletezhetünk.

Mit nevezünk kalandprogramnak?

A legegyszerűbb kalandprogram nem más, mint egy utazási prospektus vagy egy igen kifejező térkép. A játékos egy izgalmas helyszínre jut — pl. egy félhomályos föld alatti börtönbe, egy ködös, sötét erdőbe vagy egy kísértetjárta

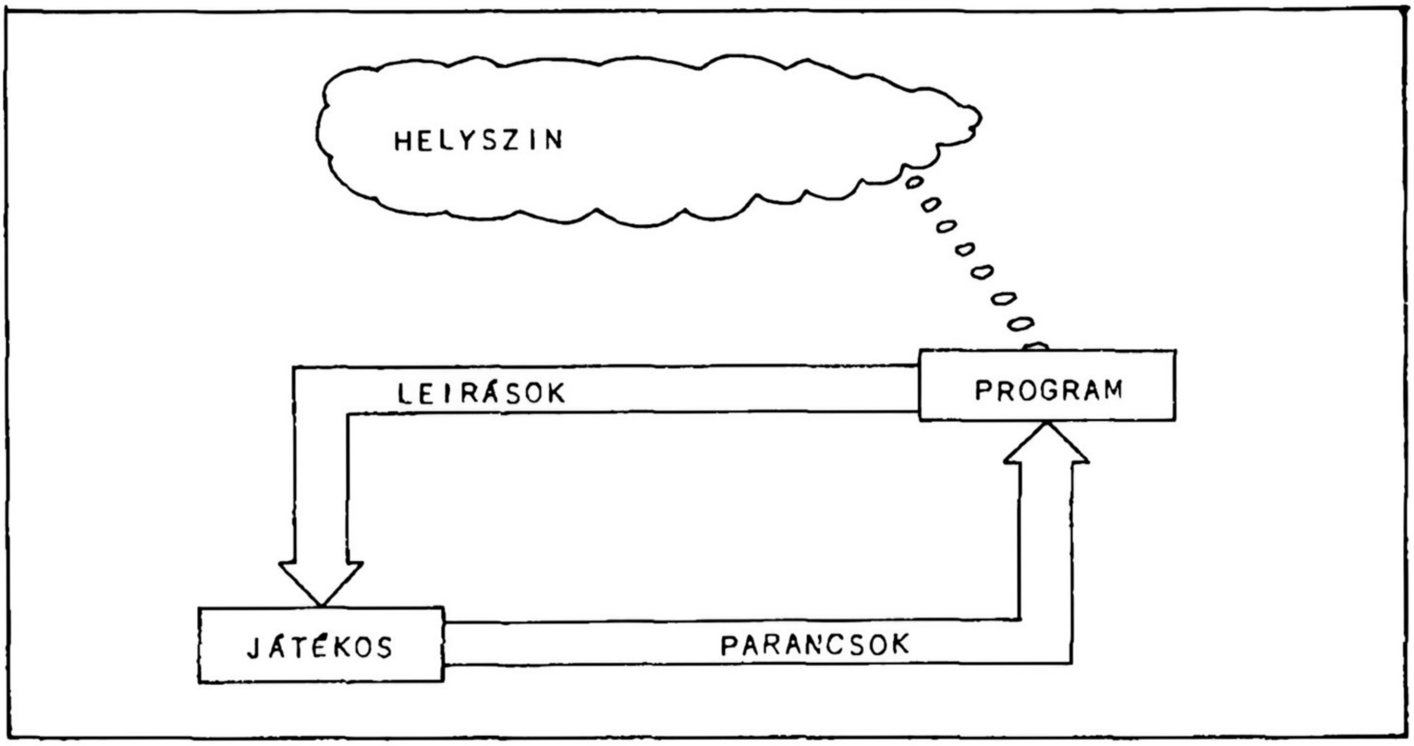
```
EGY NYILT MEZŐN ÁLL
ITT EGY DRÁGAKŐ!
EGY DÜHÖDT BIKA PATÁJÁVAL A FÖLDET
KAPÁLJA, SZINTE FELROBBAN!

●VEDD FEL A DRÁGAKÖVET!
RENDBEN

●ÉSZAK
ITT ÁLL EGY KIS ISTÁLLÓ. RÉGI, ÓSDI.

●LÉPJ BE!
A KAPU ZÁRVA VAN!
```

1-1. abra. Egy kepzeletbeli kalandprogram mintafutasa



1-2. abra. A kalandprogram es a jatekos kapcsolata

kastélyba, és elindul. A számítógép röviden leírja a helyszínt. Az újabb programok le is rajzolják, de maradjunk az egyszerű megoldásnál! A helyszín méretét — a helyzetek vagy helyiségek számával kifejezve — elsősorban a számítógép memóriája korlátozza.

A kalandprogram leíró funkcióján túl, a játékos beszélget a programmal, megváltoztatja a szimulált környezetet. A játékos egy- vagy kétszavas parancsokat ad ki, ezeket a programban egy egyszerű szótár azonosítja (1-1. ábra). Parancsok segítségével a játékos mozoghat, ajtókat nyithat-zárhat, tárgyakat tehet ide-oda, vagy más módon „léphet kölcsönhatásba" a helyszínnel. A játékosnak a játék folyamán különféle értékes tárgyakat, „kincseket" kell megtalálnia és megőriznie, amelyek el vannak rejtve ebben a mesterséges világban. Számos akadállyal kell megküzdenie; hatalmas lények: óriások, sárkányok és pókok állják útját, vagy szokványosabb ellenfelekkel, pl. portyázó hunokkal kell harcolnia. A játékos meg is halhat ebben a kitalált világban: általában fel is támadhat, folytathatja a játékot, de hatalmas pontveszteség árán. A játék igazában csak akkor ér véget, ha a játékos legyőzte ellenfeleit és megtalálta a kincseket.

Az 1-2. ábrán a klasszikus kalandprogramokra jellemző ember–gép párbeszédet láthatjuk. Jegyezzük meg, hogy a program nem ért meg mindent, de minél értelmesebb a parancsok értelmezője, annál jobb a program!

Mire van szükség?

A kalandprogram egyike a hivatásos programozás utolsó bástyáinak. A lelkes amatőr programozók már megalkották űrháborús programjaikat; egy kis segítséggel mi is készíthetünk kalandprogramokat. Ne féljünk ettől a feladattól, bárki tud tervezni labirintust és izgalmas kalandot meglepő lényekkel, pikkelyes szörnyekkel!

Mi minden kell ehhez? Nos, először is egy személyi számítógép! A könyv írásakor feltételeztük, hogy az Olvasó hozzáférhet a Radio Shack TRS–80 Model I vagy III géphez, ill. a HT–1080Z iskola-számítógéphez (mindegyik 16 kbyte RAM). Ezek a típusok Microsoft BASIC-kel működnek, könyvünk programjai is ebben a BASIC változatban íródtak. A könyvben ismertetett alapelvek azonban igazak más, BASIC-ben programozható személyi számítógépekre is, és a programok kisebb módosításokkal más gépeken is futtathatók.

Feltételezzük ezenkívül, hogy a programok tárolására rendelkezésre áll egy kazettás magnetofon (nem szükséges mágneslemez). A könyv azoknak az olvasóknak szól, akiknek a legkisebb működőképes rendszere van, akik nem engedhetik meg maguknak az összes kiegészítő berendezést. Mit érhetünk el egy 16 kbyte-os géppel és mágnesszalaggal? Hamarosan meglátjuk!

Mi kell még? Fantázia — sok! Valószínű, hogy általában sokkalta nagyobb a képzelőerőnk, mint azt hinnénk — csak elő kell csalogatni. A kalandprogramozás háromnegyed része abból áll, hogy bizarr és váratlan helyszínleírásokat találunk ki, rettenetes lényekkel és furcsa ellenfelekkel tűzdelve. Jó, ha elolvassuk J. R. R. Tolkien, Anne McCaffrey és C. S. Lewis könyveit. Olvassunk mitológiai történeteket és régi képes naptárakat; meglepődünk, mennyi ötletre lelhetünk!

Végül, jó példákra van szükség! A könyvben megtaláljuk őket. Az említett kalandprogram, a *Kardhalak és kincsek,* teljes listáját közöljük. Minden fejezet a program elkészítésének egy-egy részletét tartalmazza és néhány lehetőséget is ismertet, amit az Olvasó megvalósíthat. Biztos, hogy ez a bevezető jellegű program csak kiinduló része az Olvasó saját, jóval sikerültebb kalandprogramjainak.

Mit lehet elsajátítani?

A játék öröm, de az élet nemcsak öröm és játék! A gyakorlat végére az Olvasó elsajátíthat néhány speciális programozási technikát úgy, hogy közben észrevétlenül sokat tanul. Először is az Olvasó megismeri a *strukturált programozás* csodáit. Ne ijedjünk meg, mindössze arról van szó, hogy tapasztaljuk majd, milyen nagyszerű dolog, ha egy hosszú programban meg tudunk találni egy szubrutint, anélkül, hogy soronként szétszednénk a programot! Mire a könyv

végére ér az Olvasó, azt kívánja, bárcsak minden saját használatra írt program strukturált lenne! A dolog könnyebb, mint hinnénk.*

Néhány módszert elleshetünk azzal kapcsolatban is, hogy *hogyan használjuk fel gazdaságosan a tárat.* A kalandprogramok szinte falják a tárat. Falják a hosszú leíró szövegek, a szótárak listái és a térképek táblázatai. No persze emlékeznünk kell, hogy a legrégebbi programokat még a nagy számítógépekre írták! A tár nem is okoz gondot, ha van lemez, sőt fölösleges lemezkapacitás. Mi azonban egy mágnesszalagos, 16 K-s TRS–80 géppel dolgozunk, meg kell tehát tanulnunk úgy takarékoskodni, hogy célunkat azért elérjük.

Végül még valami, amiről sokat hallunk majd — ez az *ember–gép kapcsolat.* Mit is jelent ez? Azt, hogy mennyire érti meg a program a beírt szöveget és milyen jól reagál rá. Megtanuljuk, hogyan kelthetjük azt a benyomást, mintha egyszerű gépünk sokkal okosabb lenne, mint amilyen a valóságban. Azt is elsajátíthatjuk, hogyan alakítsuk úgy a programot, hogy javítsa a helyszín és a játékos közötti kapcsolatot, hogyan erősítse meg azt az érzést, hogy a játékos valóban egy útvesztőben bolyong, tényleg eltévedt a sivatagban, menekül az égő erdőből, vagy pedig egy Marsbeli kupolában rejtőzködik.

Talán sikerült felcsigázni az Olvasó fantáziáját, játékszenvedélyét. Ragadjunk papírt és ceruzát, kapcsoljuk be a számítógépet, és készüljünk fel a képzeletbeli ugrásra!

*Mellékesen megjegyzem, hogy engedélyükkel a „strukturált programozás" kifejezést a megszokottnál általánosabb, pusztán szakmai értelemben használom. Egyébként a formális definíciókhoz szokott egyetemi hallgatók elgondolkodhatnának, tudom-e egyáltalán, miről beszélek! (*A szerző megjegyzése*)

2. FEJEZET

Egy föld alatti színhely leképezése

A kalandprogram olyan mesterséges világgal, előre beprogramozott környezettel veszi körül a játékost, amellyel ő kölcsönhatásba léphet. Ezt a pótkörnyezetet szöveges leírások és szerteszét heverő tárgyakról szóló információk tartják fenn! A mesterséges birodalmat *helyszínnek* nevezzük.

A helyszín típusa a programozó képzeletétől függ. Az eredeti, klasszikus helyszín egy föld alatti barlang környezete, amelyben a játékosnak mitikus szörnyekkel kell küzdenie a kincsek megszerzéséért — középkori, mágikus világ, kardpárbajokkal és csodás kövekkel. A könyv példaprogramja, a *Kardhalak és kincsek* ilyen helyszínen játszódik. Persze sok más színhely is elképzelhető — hagyományosak és meghökkentők, amilyenné a programozó formálja őket! Minden az Olvasó kreativitásán múlik; bármilyen színhely lehetséges, amit egyáltalán le tudunk írni!

Vegyük pl. a következő lehetséges helyszíneket!

- A játékos egy kísértetkastély foglya. A nyikorgó ajtók mögött kísértetek és szellemek rejtőznek. Meg kell találnia a kincseket a dohos szobák mélyén.
- A játékos záróra után bentmarad az állatkertben, és ekkor az állatok kiszabadulnak a ketrecekből. Éhes oroszlánokkal, rohanó majmokkal és dühös struccokkal kell szembeszállnia, s közben folytatnia kell a keresést.
- A sötétség leple alatt be kell hatolnia egy titkos állami intézménybe, ahonnan — az épületben működő biztonsági berendezéseket „kijátszva" — néhány bizalmas iratot kell megszereznie.
- A játékos egy idegen bolygón landol. Járműve azonban leszállás közben megsérül. El kell hárítania a bolygó ellenséges lakóinak támadásait, miközben össze kell szednie a hajtómű darabjait, amelyek a földet éréskor szanaszét szóródtak.

Látható, hogy a kaland helyszíneként mindegyik megfelel, ha kielégít három követelményt. Az első maga a háttér, egy elég tágas környezet, amelyben van lehetőség a mozgásra. A második feltétel a sokféle tárgy, amelyeknek keresése és megtalálása a játék elsődleges célja. A harmadik feltétel az, hogy

számtalan akadállyal találkozzon a játékos. Egyrészt élőkkel (pl. ellenfelekkel, akikkel meg kell küzdeni), ill. élettelenekkel (pl. bezárt ajtókkal) — ezek mind-mind nehezítik és izgalmasabbá teszik a játékot.

A három követelmény közül az elsőt, az alaphelyszínt, meg kell tervezni, mielőtt a programban túlzottan előrehaladnánk; ez a fejezet éppen ezzel foglalkozik. A második és harmadik követelménnyel majd a következő fejezetben foglalkozunk.

A *Kardhalak és kincsek* föld alatti barlangja lesz a példahelyszín. Először megtanuljuk, hogy hogyan képezzük le a föld alatti helyszínt.

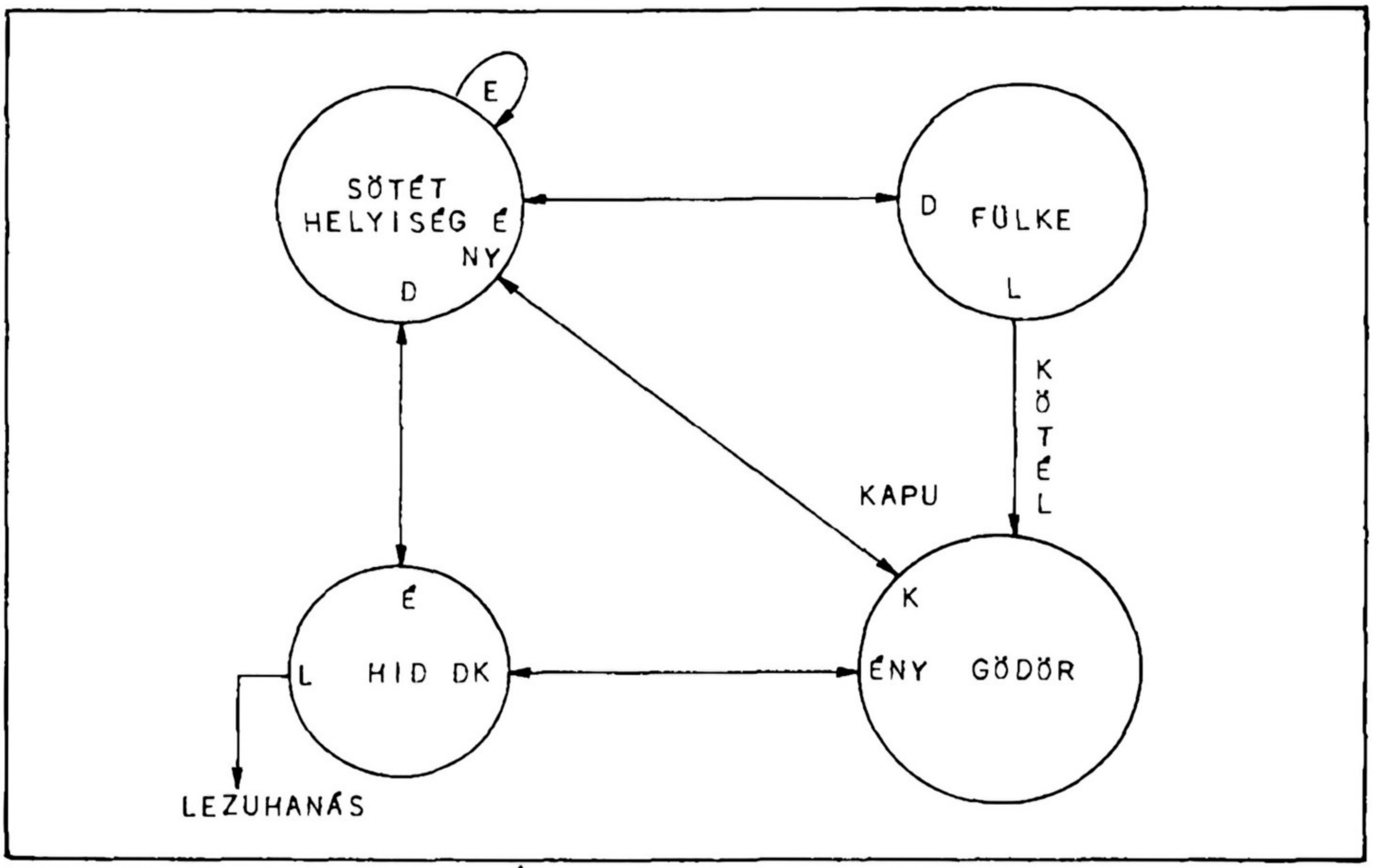
Mindenekelőtt legyen elég hely

Egy helyszínnek, példánkban egy föld alatti barlangrendszernek, azt a benyomást kell keltenie, hogy tágas. Ezt úgy érhetjük el, hogy különálló egységekre osztjuk a helyszínt. A helyszín típusától függően ez lehet egy épületbeli szoba, egy különálló barlang egy barlanglabirintusban, vagy egy erdei tisztás a sűrű erdőben. A kalandprogramban egy bekezdésnyi kifejező leírás és egy két-, háromszavas név tartozik a helyiségekhez, ill. a helyszínhez. A program közli, hogy milyen tárgyak vagy lények (ha vannak ilyenek) lehetnek a helyiségben, amikor a játékos belép. A helyiséget egyértelműen azonosítják az előre beprogramozott bejáratok és kijáratok, vagyis azok az irányok, amelyekben a játékosnak mozognia kell, hogy bejusson egy helyiségbe (vagy a helyszín egyik zárt terébe), vagy elhagyja azt.

Ha a programozónak már megvan az alapvető elképzelése, és nekilát, hogy elkészítse a helyszínt, akkor kezdetként van egy sereg helyisége, leírás és specifikáció nélkül. Ha BASIC-ben dolgozik, és csak 16 kbyte RAM áll a rendelkezésére, akkor ez komoly korlátokat jelent. Hogy a helyzetet illusztráljuk, a *Kardhalak és kincsek* nevű játékban csak húsz helyiség (ill, helyszín) van. (Kifinomult módszerek és gépi kód alkalmazásával ez a szám több, mint a kétszeresére növelhető.)

A programozónak az összekötő utakból szinte hálót kell szőnie az üres helyiségek közé, amíg minden helyiség (ez lehet egy barlang is) elfoglalja térbeli helyét a többihez viszonyítva. Más szóval ezt úgy fejezzük ki, hogy létrehozza a helyszín *térképét.*

Az összekötő utaknak ez a térképe a kalandprogramok legtöbbjében használt jelölésen, az *égtájak szerinti mozgáson* alapul. A játékos észak, dél, kelet vagy nyugat felé mozoghat, sőt átlós irányban is, pl. északkelet vagy délnyugat felé. Mehet tovább̧á felfelé vagy lefelé; mindez tízféle haladási irányt biztosít. Egy helyiség helyzetének meghatározása a többihez képest azt jelenti, hogy a programozó eldönti, mi történjen, ha a játékos kipróbálja ezeket az irányokat valamelyik helyiségben.



2-1. ábra. A helyszín rajzán használt jelölések

A 2-1. ábrán látható diagram bemutatja a négy helyiség között a lehetséges mozgásváltozatokat. Nagyméretű kockás papírt használjunk a térkép elkészítéséhez. (A szabványos 30×40 cm-es papír nagyon megfelel a célnak.) Ezután körzővel vagy inkább papírboltban kapható műanyag sablonnal rajzoljuk az összes helyiséget a papírra. A munkának ebben a korai szakaszában nem szükséges, hogy bármilyen kapcsolat legyen köztük — egyszerűen osszuk el őket egyenletesen több sorban! Pl. a *Kardhalak és kincsek* öt egyenként négyhelyiséges sorból áll. A helyiségek között hagyjunk bőven helyet a köröket összekötő vonalak számára!

Hogyan jutunk egyik helyiségből a másikba

Gondolkozzunk el egy pillanatra, mi történik, ha egyik helyiségből a másikba jutunk! Ha lakásunk valamelyik helyiségében vagyunk, és el akarunk indulni valamilyen irányban, akkor három eset lehetséges. Ezek a következők:

- Egy másik helyiségbe érünk. Ha észak felé indulunk és északra van ajtó, akkor egy új helyiségben találjuk magunkat.
- Mozoghatunk ugyanabban a helyiségben. Ha elég nagy helyiségben vagyunk, akkor mehetünk egy darabig északnak, és azt találjuk, hogy még mindig ugyanabban a helyiségben vagyunk.

• Nem jutunk el sehova. Ha keletre indulunk, falba ütközhetünk. Rendes körülmények között nem mehetünk felfelé és lefelé sem.

Most képzeljünk el egy olyan táblázatot, amelyben fel van tüntetve mind a tíz irány, amerre a játékos mozoghat! (Többet is tehetünk annál, hogy elképzeljük, nézzünk a 2-2. ábrára!) Írjuk be az üres rubrikákba azt a helyiséget, ahová a lépés eredményeként eljutunk. Más szóval a táblázatból megtudhatjuk, melyik helyiségbe jut a játékos, ha adott irányban mozdul. Ezt az eszközt *bejárási táblázatnak* nevezzük.

Most nézzük ismét a három lehetőséget! Ha a játékos az 1 számú helyiségben van, és észak felé mozogva a 2 helyiségbe jut, akkor 2-t írhatunk az ÉSZAK-hoz legközelebb eső rubrikába. Ezt egyszerűen jelölhetjük a helyszín térképén is; húzzunk vonalat az 1 helyiségtől a 2 helyiségig, és írjunk É-t az 1 helyiség körébe közvetlenül a vonal tövébe. Ez azt jelképezi, hogy ha a játékos északi irányban mozog, elhagyja a helyiséget és a 2 helyiségben köt ki. (A visszautat hasonlóan jelöljük. Egy D a vonal végén a 2 helyiségnél azt jelenti, hogy ha dél felé halad a játékos, visszatér az 1 helyiségbe.)

Lássuk a második lehetőséget! Ahhoz, hogy egy helyiség tágasnak tűnjön, bizonyos irányok ne vezessenek a kijárathoz — vég nélküli utaknak is kell lenniük. (Ez különösen jó a szabadtéri színhelyeknél, pl. erdőkben, úttalan sivatagokban stb. A későbbiekben, mikor útvesztőkkel foglalkozunk, ez szintén hasznos lesz.) Ilyen esetben írjuk be ugyanazt a helyiségszámot az irányt jelző szó alá. Ha keleti irányban mozogva a játékos az 1 helyiségben kóborol, írjunk 1-et a megfelelő helyre. A helyszíntérképen ezt önmagába visszatérő nyíllal jelöljük, amint azt a 2-1. ábrán is láthatjuk. A K betű az önmagába visszatérő nyíl belsejében azt jelzi, hogy a keleti irányú mozgás nem jelent tényleges előrejutást. A játékos ugyanabban a nagy helyiségben marad.

Végül, az utolsó lehetőség. Ha a játékos nem mozoghat egy adott irányban, pl. azért, mert egy fal állja útját, vagy nincs meg a szükséges különleges nyílás (ez a helyzet a LE és FEL esetében), ezt úgy jelöljük, hogy nullát írunk a megfelelő helyre.

Nullás számú helyiség nincs; a nulla arra figyelmezteti a programot, hogy tiltott irányban próbál a játékos mozogni. A program figyelmeztető üzenettel reagál erre, pl. kiírja a következőt: *„Arra nem mehet!”* A játékos abban a helyiségben marad, ahol eddig volt. A helyszín térképén ez a definiálatlan eset; vagyis ha egy irányt nem jelölünk, akkor az tiltott irány, amelynek nulla érték felel meg a bejárási táblázatban. Pl. a 2-1. ábrán, ha a fülkéből északra indulunk, ez nem vezet sehová, és figyelmeztető üzenetet kapunk.

Valójában létezik egy negyedik, egy különleges eset is. Mi történik pl. akkor, ha nyugati irányban haladva a játékos beleesik egy szakadékba és halálra zúzza magát a sziklák közt? A halál nem azt jelenti, hogy egy helyiségbe értünk, és többet jelent, mint vég nélküli mozgást vagy tiltott irányú mozgást. Később majd látjuk, hogy az ilyen halálesethez hozzárendelhetünk egy kitüntetett számot (nem helyiséget jelölő számot, a helyszínben szereplő helyiségek szá-

BEJÁRÁSI TÁBLÁZAT

JELENLEGI HELYISÉG	ÉSZAK	ÉK	KELET	DK	DÉL	DNY	NYUGAT	ÉNY	FEL	LE
1. HELYISÉG	2	2	1	3	0	0	1	1	0	0
2. HELYISÉG	7	9	5	4	1	1	0	0	0	0
3. HELYISÉG	5	0	9	9	0	7	1	1	0	4
4. HELYISÉG	11	0	0	0	5	0	8	2	3	0
5. HELYISÉG	4	6	6	0	3	0	2	0	6	0
6. HELYISÉG	10	0	14	0	0	5	5	7	0	5

2-2. ábra. Példa a bejárási táblázatra
A célhelyiségek irány szerint rendezettek

JELENLEGI HELYISÉG	É	ÉK	K	DK	D	DNY	NY	ÉNY	F	L	FELTÉTELEZETT
1	1	2	2	1	1	1	1	1	0	3	9
2	2	2	2	2	2	1	1	2	0	8	9
3	0	0	4	10	0	0	0	0	1	0	8
4	0	5	0	0	11	0	3	0	0	0	4
5	0	0	0	0	0	4	0	0	0	0	5
6	0	0	0	12	0	0	0	0	0	23	3
7	0	0	0	14	0	0	0	0	0	0	3
8	0	0	0	0	14	0	0	0	2	0	8
9	9	0	16	15	9	0	0	9	0	0	7

10	23	23	23	16	17	17	17	3	0	17	4
11	4	0	0	0	0	0	0	0	0	0	0
12	0	0	13	0	18	0	0	6	0	0	7
13	0	0	0	0	0	0	12	0	0	0	6
14	8	0	0	0	19	0	0	7	0	19	4
15	15	0	15	0	15	16	9	0	0	0	0
16	15	16	16	0	16	0	10	9	0	0	1
17	18	18	18	18	18	18	18	18	18	18	0
18	12	19	0	0	0	0	0	0	0	0	0
19	14	0	0	0	0	18	0	0	14	20	9
20	22	22	22	22	22	22	22	22	19	22	8

2-3. ábra. A KARDHALAK ÉS KINCSEK teljes bejárási táblázata

mánál nagyobb számot), amely jelzés a program számára. Ha a program ezzel a számmal találkozik a bejárási táblázatban, tudja, hogy ez a játékos (a bumfordi!) halálát jelenti, akit fel kell „éleszteni", ha folytatni akarjuk a játékot. Pl. a *Kardhalak és kincsek*-ben a legnagyobb helyiségszám 20; a 22-es jelenti a tűzhalált, a 23-as a lezuhanásos halált. Van hely „további" halálnemekre.

Egy bonyolult rejtély készítése

A 2-3. ábrán a *Kardhalak és kincsek* teljes bejárási táblázata látható. Amikor ténylegesen sor kerül a BASIC program megírására, ez a táblázat DATA utasítások sorozatának formájában jelenik meg — soronként egy DATA utasítás. Ezért olyan hasznos, ha az Olvasó is elkészít egy ehhez hasonló táblázatot, annyi sorral, ahány helyiség a saját helyszínén van, és az elmozdulások számára tíz oszloppal.

Várjunk csak egy percre! Nem arról volt szó, hogy csak tíz oszlop van, egy-egy a lehetséges mozgási irányok számára? Mit keres akkor itt a tizenegyedik, a „Feltételezett" feliratú? Hát, majd látni fogjuk, amikor a mozgásparancs-értelmezés feladatával birkózunk, hogy bizonyos esetekben a játékos nem mondja meg érthetően, hogy mit is akar. Mi történjen, ha a titokzatos „MENJ BE" vagy „UGORJ" parancsot adja?Milyen irányt ért ezen? Ilyen esetekben előre ki kell választani a tíz lehetséges irány egyikét, amelyiket ez a fajta parancs jelenteni fog. A tizenegyedik alkotóelemet nevezzük *feltételezett iránynak!*

A tizenegyedik oszlopba nem egy helyiség számát írjuk. Ide egy irányt jelölő szám kerül a 0-tól 9-ig, ahol a 0 az északot jelenti, 1 az északkeletet és így tovább. Ránk van bízva, hogy a játékos helyébe képzelve magunkat kitaláljuk, mit jelent legvalószínűbben egy határozatlan parancs.

Az 1-es helyiség közepén pl. egy lyuk tátong. Ha a játékos azt a parancsot adja, hogy „UGORJ", a legvalószínűbb feltételezett irány a „LE", más néven a 9-es irány. Majd jóval később látjuk meg, hogyan működik ez részleteiben.

Vegyük észre, hogy nem kell feltétlenül ismerni a helyszín összes helyiségének leírását ahhoz, hogy összekössük őket egymással, vagyis hogy hálót szőjünk közéjük! Hasznosnak bizonyulhat azonban, és ezért ajánlatos elolvasni e fejezetnek a helyiségek leírásával foglalkozó részét. Ez a rész a háló elemi részeivel foglalkozik — a struktúrával és a kapcsolatokkal —, ezek nagyban hozzájárulnak ahhoz, hogy érdekes és bonyolult rejtvény jöjjön létre.

Négy alkotóeleme van egy jó labirintus-helyszínnek: a „támaszponthelyiség", az *útszűkületek,* az útvesztők és az akadályok. Nézzük őket egyenként!

Elsőként a helyszín *támaszponthelyiségét* teremtsük meg! Ez rendszerint elkülönül a többi helyiségtől, vagy a helyszín zömét alkotó helyiségeken kívül fekszik. Többféle rendeltetése van. A program úgy kezdődik, hogy a játékos a támaszpontján van; ez a kalandozás kiindulópontja. A támaszpont egyben

menhely is, és a kincsek biztonságos tárolója. A program annak alapján pontozza a játékost, hogy mennyi kincset sikerül a játékosnak a támaszpontjára vinnie. Ez a helyiség jelenthet egy tábort az ellenséges dzsungelben, űrhajót egy ismeretlen bolygón, vagy mélytengeri kutatóhajót egy víz alatti helyszínen. A *Kardhalak és kincsek*-ben egy sziklamélyedés a támaszpont, ide a játékos, egy vakmerő archeológus, föld alatti járaton át jutott be. A támaszponthelyiség az 1-es helyiség.

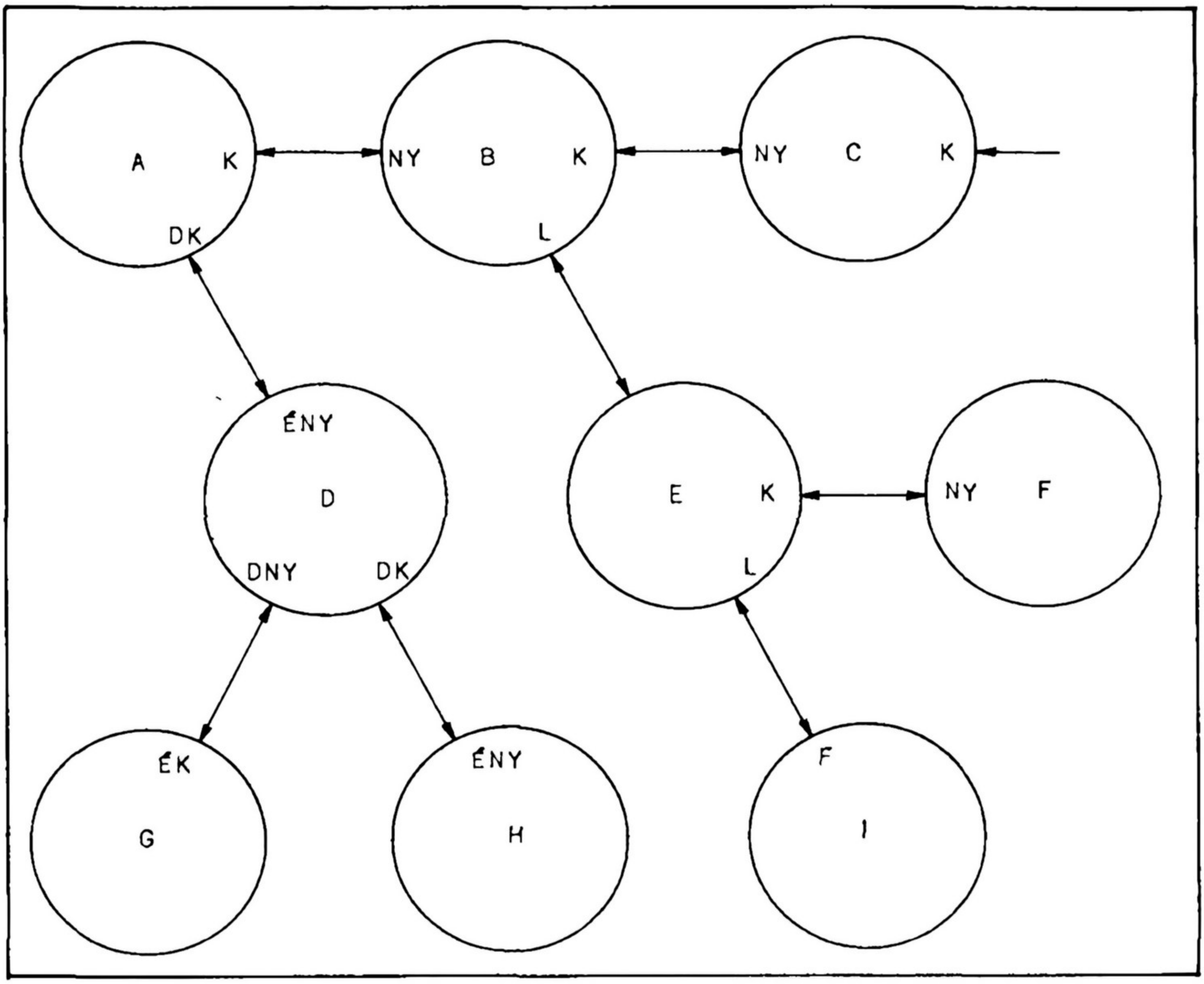
A támaszpont a helyiségekből álló bonyolult hálózat egyik bejárata. Általában legalább még egy helyiség létezik, amelyen át a legtöbb helyiség elérhető. A *Kardhalak és kincsek*-ben a 2-es helyiség romos területet jelöl, sziklás talajba beágyazott acélráccsal — nyilvánvalóan ez egy másik bejárat a barlangrendszerbe. A játékos szabadon mozoghat az 1-es és 2-es helyiségek között, de a mozgás javarészt a lenti helyiségekre esik.

A bejárási táblázatból látható, hogy az 1-es és 2-es helyiségekben a mozgási lehetőségek főleg *önmagukba visszatérő nyilak.* Ennek az a célja, hogy a nagy méret látszatát keltsék. A játékos hosszú utat tehet meg a 2-es helyiségben, és még mindig a romok között tévelyeg. Csak néhány meghatározott irány vezet a mélyedéshez, az 1-es helyiségben levő támaszpontra. Ez nyílt helyszíneken jól használható eszköz. Azt is figyeljük meg, hogy szinte nincs tiltott irány (kivéve a „fel" irányt), mivel egy tágas, nyílt helyszínen egy személy mozgását semmi sem korlátozza. A nulla érték sokkal természetesebben használható zárt környezetben, pl. egy épületben vagy egymáshoz kapcsolódó barlangokban.

Az útszűkületelv

Egy adott helyszínen a feszültség úgy fokozható leginkább, ha korlátozzuk a játékos mozgási lehetőségét. Más szóval rákényszerítjük, hogy a lehető leghatékonyabban tevékenykedjen, hogy megtalálja a keresett helyiséget. Ennek egyik módja az, hogy a háló részeit ágakból építjük fel, ahogy ez a 2-4. ábrán látható. Ha a játékos eljutott az A helyiségbe, el kell döntenie, hogy melyik utat deríti fel. Ha választott, akkor vissza kell térnie az A helyiségbe, mielőtt hozzáláthat egy másik út felderítéséhez. Jó példa erre az *útszűkület,* ott ui. kényszerítjük a játékost, hogy áthaladjon egy adott helyiségen. Ha több útszűkületet helyezünk el, a végeredmény egy részekre bontott helyszín, amelyben mindig fennáll annak a lehetősége, hogy a játékos új, fel nem derített részekre bukkan.

Az útszűkület létrehozásának másik módja az egyirányú utak kijelölése. Ebben a helyzetben az A-ból B-be mozgó játékos nem juthat vissza A-ba ugyanazon az úton — kerülőútra kényszerül. Ilyen esetben a helyszín térképén elhagyjuk a visszafelé vezető irány jelzését a B-be vezető úton. A bejárási táblázatban nullát írunk a visszafelé mutató érték helyére — esetleg egy teljesen



2-4. ábra. Példa útszűkületet eredményező helyiségelrendezésre

más helyiséget. Ezt megtehetjük, amíg nincs közvetlen visszavezető út az A helyiségbe.

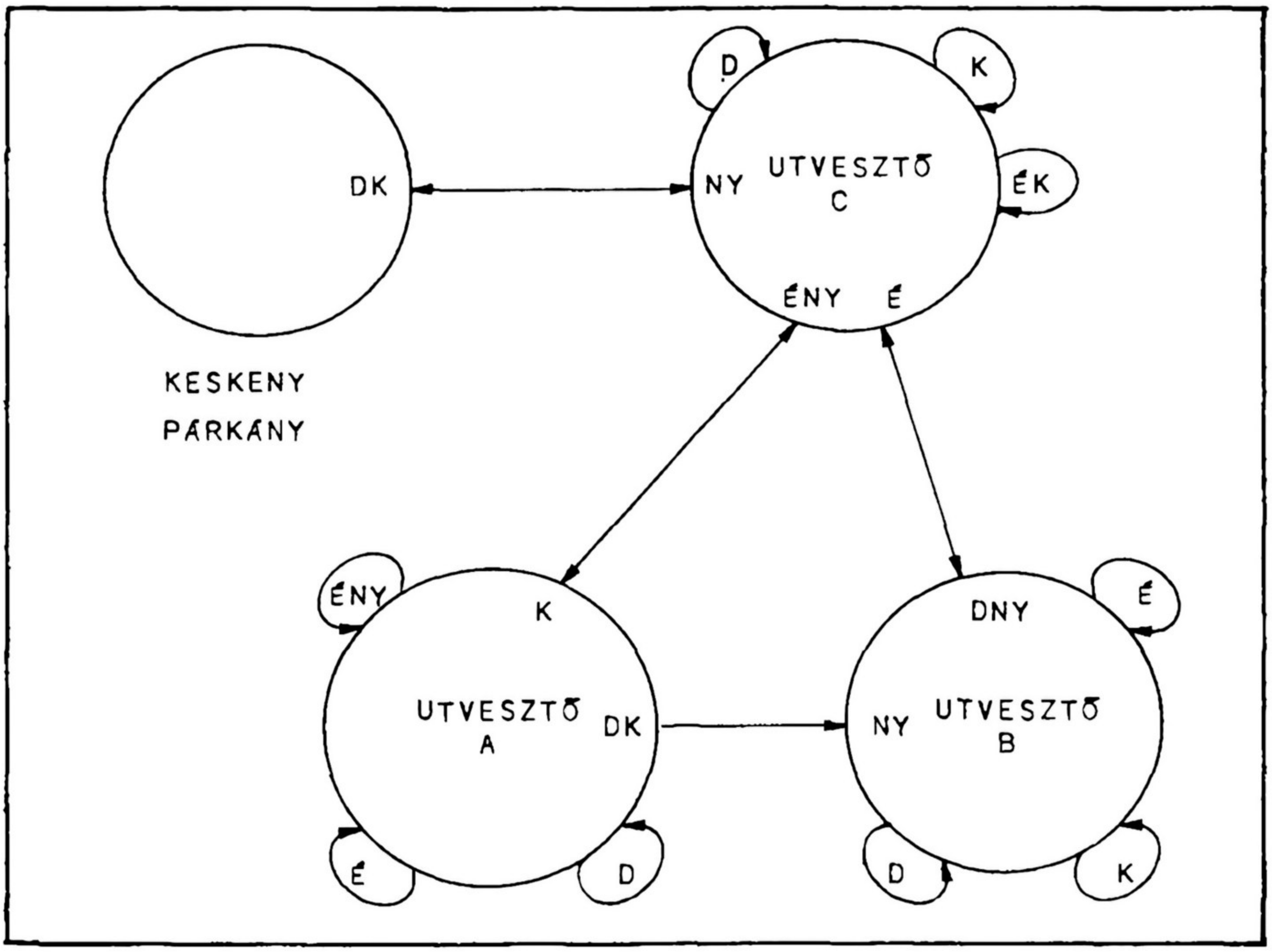
Az ilyen helyzetre többféle magyarázat is adható a helyszínleírásban. Talán az A helyiség jóval magasabban van a B-nél, és a játékos leugorhat a B-be, de nem tud visszamászni. Esetleg az A és a B egy áruházi mozgólépcső két végpontján helyezkedik el. A *Kardhalak és kincsek*-ben van egy sziklapárkány és a mélyben egy folyó. Bele lehet ugrani a folyóba, sőt ezt az ugrást túl lehet élni, de sohasem lehet ismét feljutni a párkányra. A lényeg: csak egyirányú mozgás végezhető!

Az útszűkületnek van egy rejtettebb fajtája is: az *álcázott ösvény*. Ennek a módszernek az „ereje" az Olvasó leíróképességében van. Rendszerint a helyiséget leíró bekezdés elárulja, hogy milyen átjárók vagy utak láthatók közvetlenül a játékos körül. Ezek a leírt kapuk megfelelnek a bejárási táblázat átjáróinak. Az azonban nincs előírva, hogy mindent el kell mondani. Pl. besétálhat a játékos az északi irányban található bokrok közé, és ott egy olyan tisztást (új helyiséget) találhat, amely nem látható. És — talán az a délre fekvő

fal nem is olyan szilárd, mint amilyennek látszik! Lehet egy vízesés keletre — de láss csodát, mi történik, ha egyenesen belesétálsz a vízbe!? A lényeg az, hogy célozzunk lehetséges titkos kapukra (váratlan fordulatokra), és reméljük, hogy minél tovább bizonytalanságban tarthatjuk a játékost.

Eltévedés az útvesztőben

Aligha nevezhető teljesnek egy kalandprogram útvesztő nélkül. Természetesen, bizonyos értelemben az egész helyszín útvesztőnek tekinthető, de most használjuk szűkebb értelemben ezt a szót! A kalandprogram útvesztője azonos vagy nagyon hasonló leírású helyiségek olyan halmaza, ahol a játékos könnyen és reménytelenül eltévedhet. Mikor a játékos egy ilyen helyiségben van, választhat, melyik irányban halad. Ha a helyiség leírása pl. ilyen: „ELTÉVEDT EGY ÚTVESZTŐBEN", elgondolkodhat: „Most ugyanabban az útvesztőben vagyok, esetleg egy másikba kerültem?" Nincs szükség túl sok helyiségből álló útvesztőre ahhoz, hogy hatékony csapdát állítsunk a játékosnak.



2-5. abra. Harom helyisegbol allo, egy kijaratu utveszto

A 2-5. ábrán egy kisméretű, három helyiségből álló útvesztőt látunk, azonosat azzal, ami a *Kardhalak és kincsek*-ben szerepel. Az útvesztő helyiségeit három alapvető dolog jellemzi. Az elsőt már említettük: a megkülönböztethetetlen leírás. A második jellemzőt nem kell útvesztőkre korlátozni: a bejárási táblázatban ne az ellentett irányt használjuk. Figyeljük meg pl., hogy a játékos úgy jut az A útvesztőből a C útvesztőbe, hogy kelet felé mozog. Vissza azonban északnyugat felé haladva jut, nem nyugat felé. Ezt a módszert következetesen végigvisszük az útvesztőben oly módon, hogy a játékos soha se legyen biztos abban, hogy hogyan is jut el egy adott helyiségbe. El tudjuk képzelni, milyen a balsiker érzése, ugye?

A harmadik jellemző a *hurkok* széles körű használata. Figyeljük meg, hogy az útvesztő minden helyiségében három hurok van! Nagy a valószínűsége, hogy a játékos több lépést tesz különböző irányokban, és azt hiszi, hogy teljesen új helyiségekbe jutott — miközben valójában egy helyen toporog! A három alkotóelem hatékony használatával elérhetjük, hogy a játékos teljesen megzavarodik, anélkül, hogy túl sok helyiség lenne.

A tapasztalt játékos úgy jut ki az útvesztőből, hogy leejt egy nála levő tárgyat, mielőtt elmozdulna. Ha a helyiség leírásában szerepel az elejtett tárgy, akkor tudja, hogy még ugyanabban a helyiségben van, nem hagyta el. Ily módon lényegében minden helyiséget megjelölhet egy-egy tárggyal (amíg van elejthető tárgy nála), és megtalálhatja a kivezető utat. Az átlagos kalandozó azonban szinte sohasem próbál így eljárni, ha először van egy útvesztőben. Ehelyett vaktában szaladgál ide-oda, össze-vissza futkos, amíg puszta szerencsével kijut!

Azok a bosszantó akadályok!

Nézzük egy percre a *Kardhalak és kincsek* teljes térképét minden útszűkületével és cirádájával (2-6. ábra)! Most tekintsünk el a helyiségek nevétől (később megmagyarázzuk őket), és figyeljünk a háló többi jellemzőjére!

Emlékezzünk rá, mit is mondtunk azokról az irányokról, amelyek tűzhalálhoz vagy lezuhanásos halálhoz vezetnek! Három ilyen helyiség van a *Kardhalak és kincsek*-ben, ahogy ez rögtön kiderül a bejárási térképről. A 22 jelenti a tűzhalált és a 23 a lezuhanásos halált. A 6-os, 10-es és 20-as helyiségekben szerepel a két érték valamelyike a mozgás eredményeként. Most nézzük, hogyan jelöljük ezt a helyszín térképén! A mozgás irányát jelző nyíl a halált jellemző szóra mutat: TŰZ vagy LEZUHANÁS.

Mérjük fel, hogy milyen értelemben jelent ez akadályt! Nem kell közölni a játékossal ui., hogy végzetes irányban halad, elég egy célzás. Csak kövesse el azt az ostoba, vakmerő tévedést, hogy kipróbálja! A 10-es helyiség jó példa. A leírásból úgy tűnik, hogy a nyugatra fekvő folyó és a keletre található szakadék egyformán végzetes! Ha a játékos megpróbálkozik a folyóval, a helyszín-

nek egy egészen új részén találja magát! Ez arra feltevésre csábítja, hogy talán további kaland vár rá a szakadékban. Téved, halálosan téved!

A halál nem végleges a kalandprogramban; a játékos feléleszthető, de általában pontot veszít. A támaszponton éled fel, távol attól, ahol meghalt. Vagyis az utat újra be kell járnia!

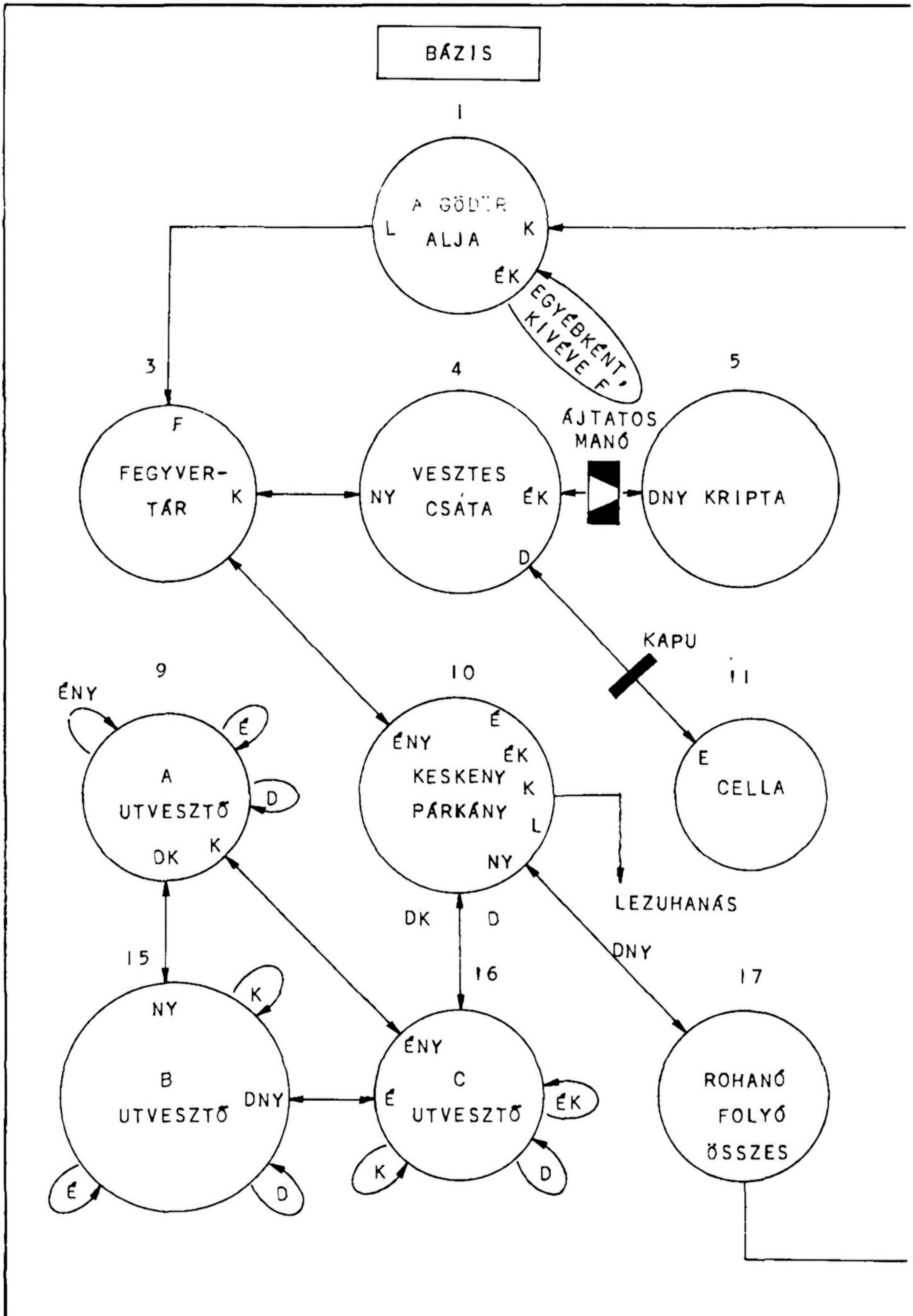
Létezik a tárgyaknak egy sajátos osztálya: ezek az *akadályok*, amelyeknek egyetlen célja, hogy változatos módon gátolják a kalandozó előrehaladását. A kalandprogram az akadályok állapotáról aktuális listát vezet, jelezve, hogy melyik utat zárják el, és ott vannak-e még. Ezt a listát egyszerűen akadálylistának nevezzük, ami nem más, mint egy változó vektor, amit a BASIC program vezet.

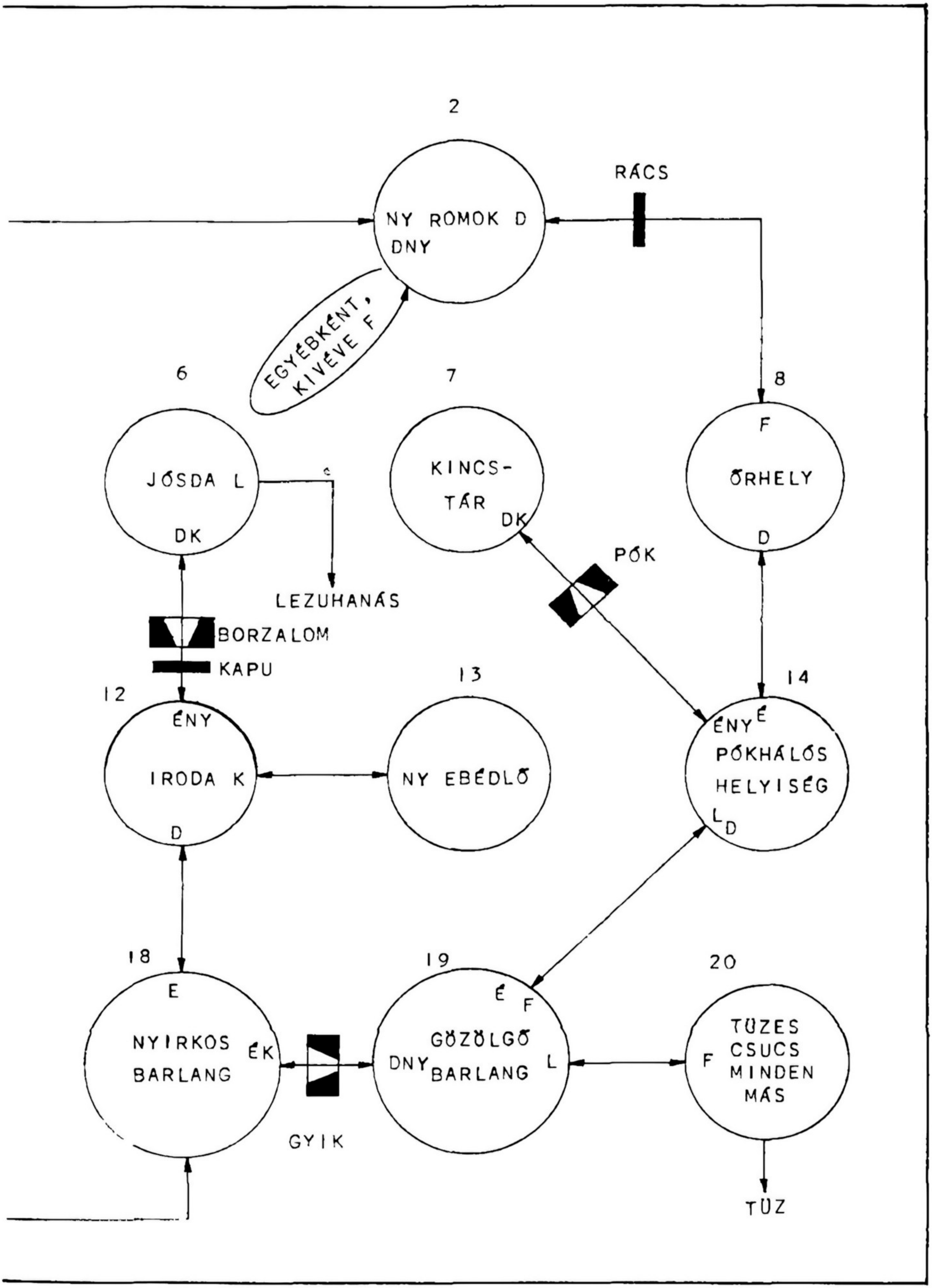
Az akadályoknak két fő típusa van. Léteznek élő akadályok, amelyek rendszerint szörnyek vagy valamilyen vadállatok, és egy adott átjárót őriznek. Általában harcban, meghatározott fegyverrel lehet legyőzni őket. Az akadályok másik csoportját az élettelen tárgyak, pl. kapuk vagy acélrácsok alkotják. Ezeket az akadályokat úgy távolíthatjuk el, hogy kulccsal kinyitjuk vagy kilakatoljuk őket. Ezeknek az akadályoknak a balsiker érzését növelő értéke abban áll, hogy különleges tárgyakkal — fegyver, kulcs — lehet elbánni velük. Ha pl. a mindennél fontosabb kulcs valahol a színhely mélyén van, sok helyiséget és kincset nem láthat a játékos addig, amíg meg nem találja a kulcsot. Ezekkel az eszközökkel a játékidő növelhető, az izgalom fokozható.

Nézzük először az élettelen akadályokat, mivel ezeket a legnehezebb kezelni. A *Kardhalak és kincsek* térképén három ilyen akadályt figyelhetünk meg; acélrács választja el a 2-es és 8-as helyiséget, kapu áll a 6-os és 12-es helyiség között, és egy másik kapu osztja ketté a 4-es és 11-es helyiséget.

Közönséges kapu esetén valójában három állapot lehetséges. A kapu vagy a rács be lehet csukva és zárva, becsukva, de be nem zárva, vagy nyitva és be nem zárva. Egyszerűbbé teszi az ilyen fajtájú akadályok kezelését, ha a lehetőségek számát kettőre csökkentjük. Más szóval egy akadályon át lehet haladni, vagy nem, egyszerű vagy-vagy feladat. Ezt úgy ábrázolhatjuk a helyszínrajzon, hogy a kapukat és ajtókat úgy kezeljük, mintha a két lehetséges helyzet egyikében lennének. Programozási szempontból A jelenthet becsukott, bezárt , nem járható kaput; B jelenthet be nem zárt nyitott és járható kaput.

Ez az egyszerűsítés csak kevéssé befolyásolja a helyszín valósághűségét, de jelentősen csökkenti az akadálykezelés bonyolultságát. Ha a játékos olyan irányban próbál haladni, amelyet egy A állapotú kapu zár le, a következő üzenet tartóztatja fel: „A KAPU BE VAN ZÁRVA ÉS LE VAN LAKATOLVA". Ha úgy próbálja kinyitni és kitárni, hogy nincs kulcsa, a program azt válaszolja, hogy „NINCS KULCSA!" Ha nála van a kulcs, a program B-re változtatja a kapu állapotát és közli a játékossal, hogy „A KAPU CSIKOROGVA KITÁRUL". Ezután, ebben az irányban szabadon haladhatunk — az ajtó már nem akadály, mivel B állapotban van. A játékos bezárhatja a kaput, és visszaállíthatja az akadály állapotát A-ra. A következő üzenet közli ezt:


BÁZIS
1
A GÖDÖR ALJA
L
K
ÉK
EGYÉBKÉNT, KIVÉVE F
3
F
FEGYVER-TÁR
K
4
NY
VESZTES CSÁTA
ÉK
D
ÁJTATOS MANÓ
5
DNY KRIPTA
KAPU
9
ÉNY
É
D
A UTVESZTŐ
DK
K
10
É
ÉNY
ÉK
KESKENY PÁRKÁNY
K
L
NY
LEZUHANÁS
11
E
CELLA
DK
D
DNY
15
NY
K
B UTVESZTŐ
DNY
É
D
16
ÉNY
C UTVESZTŐ
É
ÉK
K
D
17
ROHANÓ FOLYÓ ÖSSZES



2-6. ábra. A KARDHALAK ÉS KINCSEK teljes helyszínének térképe

„A KAPU BEZÁRUL, ÉS A ZÁR BEKATTAN", és ezentúl ebbe az irányba nem mehetünk.

Nyilvánvaló, hogy az ilyen akadályt nem kezelhetjük egyszerűen olyan tárgyként, amelyik egy helyiségben van. Mert pl. egy kapu két helyiségre nyílik; ha akkor nyitjuk ki a kaput, mikor az A helyiségben vagyunk, ugyanezt a kaput nyitva kell találni akkor is, ha a szomszédos B helyiségben vagyunk. Ezért minden kapu típusú élettelen akadály számára két állapotszámot kell fenntartani; egyet-egyet a két helyiségre, amit érint. A programnak egyszerre kell mindkét szám állapotát megváltoztatni, ha a kaput kinyitják vagy bezárják.

Bejegyzések az akadálylistán

Most lássuk, hogyan állítjuk elő az akadálylistát és különböző bejegyzéseit, hogy megkönnyítsék a kalandprogram akadályainak kezelését! Ez a könyvben tárgyalt fogások közül a nehezebbek közé tartozik, olvassuk tehát figyelmesen!

Minden kapuhoz az állapot–információ két külön halmaza tartozik, ez szükséges ahhoz, hogy a program helyesen szimulálja az akadályt.

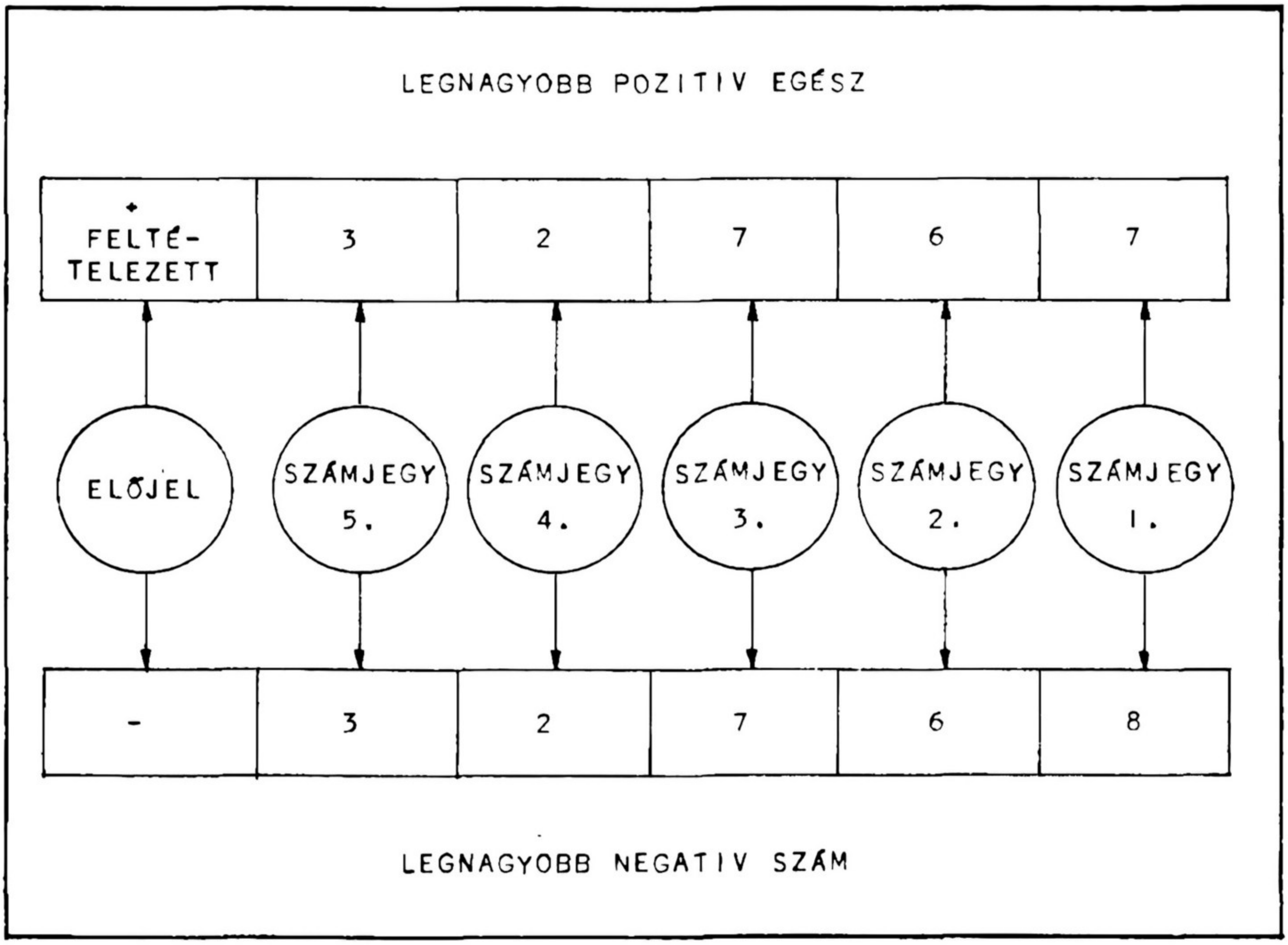
A következő információkra lesz szükség:

- Melyik helyiséget érinti? Ez a szám természetesen A az A helyiségre nézve és B a B helyiségre, ha a kapu ezt a két helyiséget választja el.
- Melyik irányt gátolja? A kapu lehet az A helyiség északi falán és a B helyiség déli falán. A 0-tól 9-ig terjedő irányt jelző szám minden helyiség esetén elárulja az akadályozott mozgásirányt.
- Milyen típusú az akadály? Ennek elsődleges célja, hogy tudassa a programmal, milyen üzenetet írjon ki, pl. azt írja-e, hogy „A KAPU NYITVA VAN", vagy hogy „A RÁCS NYITVA VAN". Terméészetesen ez a szám azonos mindkét érintett helyiségnél.
- Járható-e most az akadály? Ez az a tényleges állapotjelző, amely elárulja, hogy a kapu be van-e zárva és le van-e lakatolva, vagy sem. Ez ismét azonos mindkét érintett helyiségnél.

Mivel az akadályok állapotát jelző adatok listában vannak elhelyezve (az akadálylistában), szükség van még egy információra. Szükség van egy számra, amely megmondja a programnak, hol van a másik halmaz állapot–információja a listában. Hiszen ha egy kaput kinyitunk, nemcsak annak a helyiségnek az adatát kell megváltoztatni, amelyben a játékos tartózkodik, de a szomszédos helyiségét is. A jelenlegi helyiség állapot–információját a program könnyen megtalálja, de hol van ennek az adatblokknak a párja?

- *Hol van az állapotot jelző adat párja?* Az egyszerűség kedvéért az állapotblokkpárokat egymás mellé rakjuk az akadálylistán.

Ha azonban valamelyiket nézzük, a másik vajon pont előtte van-e vagy mögötte? Ezt egyetlen szám segítségével eldönthetjük, amely azt jelzi, hogy



2-7. ábra. Így bontható fel a Microsoft BASIC-ben használt szabványos egész a hatékony adattárolás érdekében

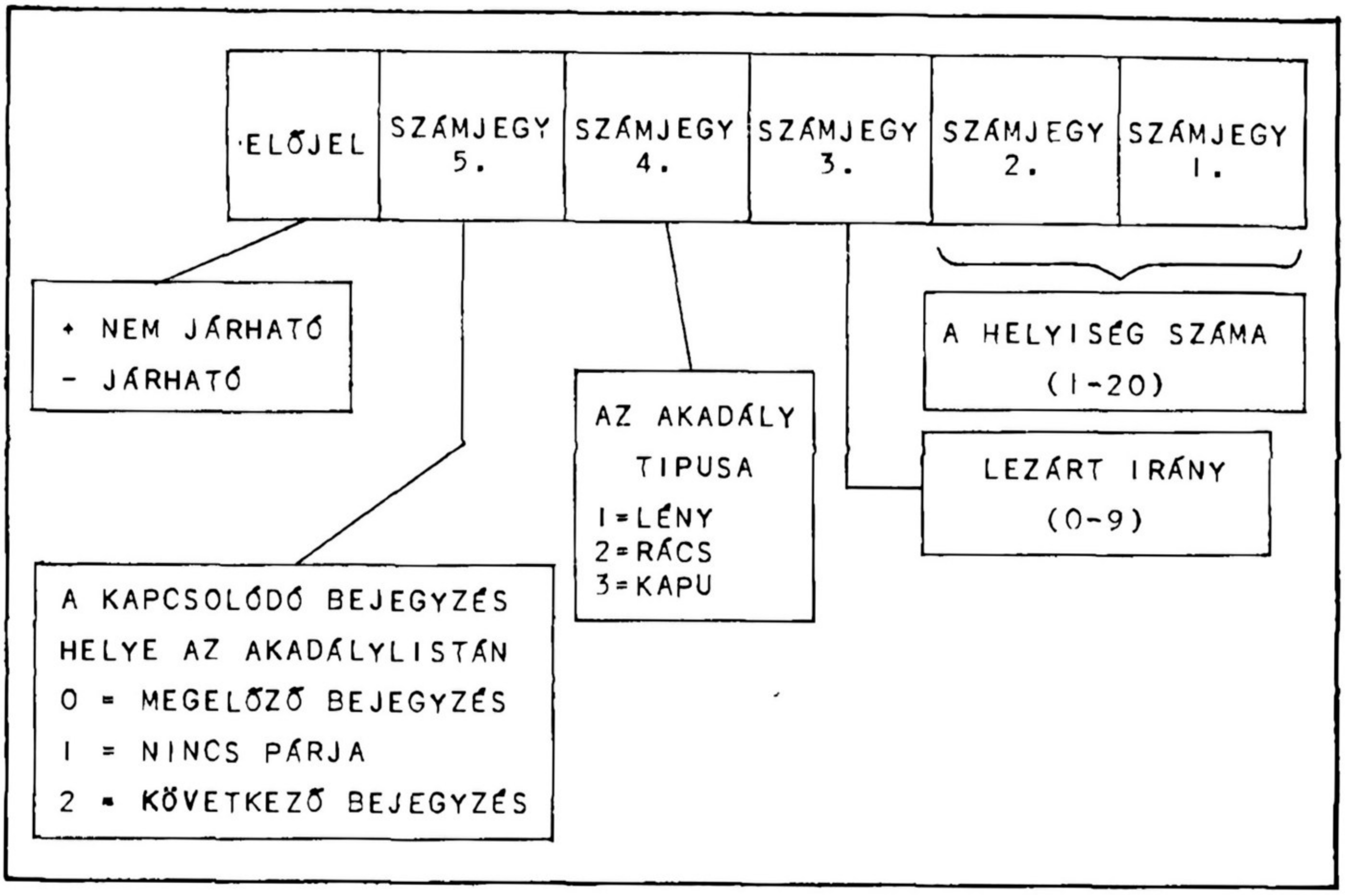
„előtte" vagy „mögötte". Ez az adat ellentétes a blokkpárjánál. A pár első tagja úgy mutat a másikra, hogy azt mutatja „mögötte"; a másik a párjára úgy, hogy „előtte".

Épphogy túljutottunk egy nehéz fogalmon, nézzük át újra és egyszerűsítsük kissé! Minden kapu típusú akadály két helyiséget érint. Mindkét helyiséghez tartozik egy bejegyzés az akadálylistán. Ez a bejegyzés valamilyen módon öt adatot tartalmaz: a helyiség számát, az irányt, amelyet az akadály blokkol, az akadály típusát, az akadály állapotát és a másik bejegyzés helyét a listában.

Az összes adat számokkal ábrázolható: kétszer öt számmal.

Szerencsére rendelkezésre áll egy egyszerűbb, tömörebb mód a számok kezeléséhez. Mind az öt viszonylag kis számot belesűríthetjük egy nagy számba. Hogyan? Úgy, hogy egy nagy egészet felbontunk sok-sok számjegyre, és egy vagy két számjegyet rendelünk az adatokhoz. Ez nagyon sokat segít abban, hogy takarékoskodjunk a tárral, és sok helyen nagyon jól jön a kalandprogramban.

A 2-7. ábra az egész szám szabványos alakját mutatja a Microsoft BASIC-ben. Egy egész −32768 és +32767 közé eshet, a határokat is beleértve. Más szóval, a legnagyobb létrehozható egész öt számjegyet tartalmaz és egy pozitív



2-8. ábra. Az egyes számjegyek jelentése az akadályok listájának egyes elemeinél

vagy negatív előjelet. Ha az ötödik számjegy legföljebb 3, és ügyelünk a negyedik számjegyre, tetszés szerint kezelhetjük az egyes számjegyeket és akármilyen jelentést adhatunk nekik, ill. bármire fölhasználhatjuk.

A 2-8. ábra bemutatja, hogyan használható a módszer (nevezzük *egészek felbontásának*) az akadálylista egy bejegyzésének összeállítására. A helyiség száma legföljebb 20 lesz (ha több lenne a programban, akkor sem valószínű, hogy a 99-et meghaladja). A lényeg az, hogy az egész szám két számjegyére van szükség a helyiség számának megadásához. Az első és a második számjegy megfelel.

Aztán szükség van a blokkolt irányra. Emlékezzünk rá, tíz irány lehetséges; az égtájakból származó nyolc irány és a le, meg a föl. Ezt a 0-tól 9-ig terjedő számsort hozzárendelhetjük és berakhatjuk a 3. számjegybe. A negyedik számjegy jelölheti az akadály típusát, ha 0-tól 9-ig önkényesen rendelünk hozzá számokat. Az 5. számjegy a bejegyzés párjára mutat, a listában (jelentse önkényesen 0, hogy a másik bejegyzés ezt megelőzi és jelentse 2 hogy mögötte van).

Végül hogyan dönthető el, hogy a kapu zárva van vagy nyitva? Legyen az előjel az állapotjelző. Ha pozitív, az akadály nem járható (zárva van); ha negatív, akkor járható (nyitva van). Ügyes ugye?

Az akadálylista — legalábbis ami az élettelen, kapu típusú akadályokat illeti — egész számpárok sorozatából áll, és mindegyik pár egy adott kaput jelent. A 2-9. ábrán látható a *Kardhalak és kincsek* akadálylistájának ez a része,

1. BEJEGYZÉS	22902
2. BEJEGYZÉS	2808
3. BEJEGYZÉS	23306
4. BEJEGYZÉS	3712
5. BEJEGYZÉS	23404
6. BEJEGYZÉS	3011

2-9. ábra. Az akadályok listájának eleje

abban a helyezetben, amikor minden akadály zárva van. Vegyük a fáradságot és próbáljuk megérteni az egyes számjegyek jelentését!

Például a lista első bejegyzése 22902. Az olvasó biztosan érti, hogy:

- Az érintett helyiség a 2-es.
- A blokkolt irány a 9-es (LE).
- Az akadály 2-es típusú (rács).
- A bejegyzés párja ezt *követi*, ezt jelzi a 2-es.
- A rács be van csukva és be van lakatolva, ezt jelzi a pozitív szám.

Elemezzük a 2-es bejegyzést hasonló módon. Ne felejtsük, hogy ha egy számjegy értéke 0, az teljesen elmaradhat — legalábbis felületesen nézve úgy tűnhet. A 2-es bejegyzésben az ötödik számjegy értéke nulla; ezért a bejegyzés négyjegyű. Ne aggódjunk, képzeljük ötjegyű számnak egy láthatatlan vezető nullával!

Hogyan használhatja egy kalandprogram az akadálylistát az akadályok kezelésében? Tegyük fel, hogy a játékos beírja, hogy „CSUKD BE A KAPUT". A program két információval rendelkezik: tudja melyik helyiségben van a játékos (ezt mindig nyilvántartja), és tudja, hogy a játékos be akar csukni egy kaput, egy 3-as típusú akadályt. A program elkezd keresni a listában egy olyan bejegyzést, amely kielégíti ezt a két követelményt. Miután megtalálta, a program pozitívra állíthatja a bejegyzés értékét, becsukva az ajtót abban a helyiségben. Az ötödik számjegy segítségével a program megtudhatja, hol találja a másik megváltoztatandó bejegyzést. Miután megtalálta, a második bejegyzést is pozitívra állítja, és kiírja a következő üzenetet: „A KAPU BEZÁRUL ÉS A ZÁR BEKATTAN."

Természetesen ellenőrizni kell egy sor mellékes tényezőt. Mi történjen, ha a kapu már zárva van? Ezt a program a listabejegyzés előjeléből tudhatja,

és azt üzenheti: „MÁR ZÁRVA VAN!" A listabejegyzések a program látszólagos intelligenciájának kulcsai.

Még egy dolgot el kell mondani az akadálylistáról. Mivel a bejegyzéseket időről időre fel kell frissíteni, a lista nem lehet egyszerűen egy BASIC DATA utasítás számsorozata. Az eredeti értékek lehetnek DATA utasításban, de át kell tölteni őket egy változó vektorba, hogy az egyes elemeket pozitívra vagy negatívra állíthassuk, ahogy a játékos kölcsönhatásba lép a környezettel.

Vadállatok mint akadályok

Ha az eddigieket sikerült túlélni, akkor nem okoz gondot a másik típusú, az élő akadály. A színhelyet benépesítő lényeket sokkal könnyebb kezelni az akadálylistában.

Nézzük először, hogy a kapukhoz két bejegyzés tartozott az akadálylistán, mivel bizonyos értelemben két helyiséget foglaltak egyszerre! A lények ezzel szemben olyan objektumok, amelyek egy időben csak egy helyiségben vannak. Ily módon (örömök öröme) egy lényhez csak egy bejegyzés tartozik a listában. Nem kell aggódni amiatt, hogy két számot kell megváltoztatni, ha egy lény állapota változik.

Vessünk ismét egy pillantást a 2-6. ábrán látható helyszíntérképre! Azonnal látható, hogy négy akadályt jelentő lény van: egy-egy a 4-es, 6-os, 14-es és 18-as helyiségben. Pl. a játékos nem mozoghat északkelet felé a 4-es helyiségben; az Óriás Ájtatos Manó nem engedi tovább! Ha a játékos az 5-ös helyiségben van, és át akar jutni a 4-es helyiségbe, az Ájtatos Manó beengedi, mintha a játékos belopódzna a háta mögött. Ez ismét felhívja a figyelmet arra, hogy a „lény típusú" objektumok csak egyirányú akadályok. Ezért igényelnek csak egy bejegyzést a listában.

Most nézzük a *Kardhalak és kincsek* teljes akadálylistáját, a 2-10. ábrán! Megmaradt a három pár passzív akadály bejegyzése; de kiegészül négy egyszeres bejegyzéssel, lényenként eggyel-eggyel.

Az első lény bejegyzése 11104. Elemezzük, mit jelent ez az akadályra nézve. Az akadály a 4-es helyiségben van. Az 1-es irányban gátolja a haladást, ez történetesen északkelet. Az akadály 1-es típusú, ezt önkényesen „lény"-ként definiálom. A lény jelen van, elzárva az átjárót, mivel a bejegyzés értéke pozitív. És mi a helyzet az ötödik számjeggyel, amely arra volt fenntartva, hogy a második bejegyzésre mutasson? Lények esetén nincs második bejegyzés, az ötödik számjegy 1-re van állítva, ez jelenti, hogy ennél az akadálynál csak egy bejegyzés található. Később, amikor a tényleges BASIC programot elemzi az Olvasó, látni fogja, miért a 0, 1, 2 számokat választottuk az 5. számjegyhez.

A másik három egyszeres bejegyzés értelmezése hasonló. Ahogy a program keresgél a listában, rájön, hogy egy adott egyszeres bejegyzéssel megadott lényt ábrázol, amelyik egy konkrét helyiségben egy bizonyos utat őriz. Azt azonban

1	22902	LEZÁRT
2	2808	IRÁNY
3	23306	KAPU
4	3712	
5	23404	KAPU
6	3011	
7	11104	ÁJTATOS MANÓ
8	11118	GYIK
9	11714	PÓK
10	11306	BORZALOM

2-10. ábra. A KARDHALAK ÉS KINCSEK teljes akadálylistája

nem tudja eldönteni, hogy egy ájtatos manóról vagy egy pókról van-e szó; ez a megkülönböztetés egy másik listából derül ki, ahogy ezt hamarosan meglátjuk. Ezen a ponton az is nagy segítség, hogy ennyi mindent sikerült feljegyezni egy rövid számsorozattal.

Hogyan vált állapotot egy lény típusú akadály? Más szóval, hogyan válik átjárhatóvá vagy át nem járhatóvá? Ennek szokásos módja a harc. Ha a játékos rendelkezik a megfelelő fegyverrel, és a harc véletlen tényezői kedvezően alakulnak, megöli a szörnyet, és az akadály megszűnik. Ha kudarcot vall, a lény tovább őrzi az utat. A harc megvívásának módjával nem itt foglalkozunk, csak a 7. fejezetben, de a programnak az a része felelős a listabejegyzés negatívra állításáért, amely eldönti a harc kimenetelét, jelezve ezzel az akadály átjárhatóságát.

Mind az élő, mind az élettelen akadályok hatékonyan hozzájárulnak a kaland helyszínének élethűségéhez. Látni kell azonban, hogy ez a luxus „sokba kerül". Most már van egy változó vektor, amelyet karban kell tartani. Az az igazság, hogy valahányszor a játékos lépni próbál, az akadálylistán ellenőrizni kell, vajon a kiszemelt irány le van-e zárva vagy sem.

Igen lényeges programozási szempont, amely érvényesül minden bonyolult programban: a látványosság és a sebesség között kötendő kompromisszum. Lehetőség van rá, hogy további akadályokkal bővítsük a helyszínt, de ezért a feldolgozás idejének növekedésével kell fizetni. Ha ügyesek vagyunk, a késlekedés idejét leszoríthatjuk.

Tárgyak megtalálása

Egy kaland helyszíne nemcsak barlangokból, helyiségekből, ösvényekből és kapukból áll. A játékosnak különféle tárgyakat kell megtalálnia, miközben a köré festett mesterséges világban vándorol. A tárgyak azonban felhasználásuk szerint különböznek.

Egyfajta tárgy az a kulcs, amely kinyit egy ajtót (és így átjárhatóvá tesz egy akadályt). Másfajta az a kard, amely megöl egy adott fenevadat (így járhatóvá tesz egy teljesen másfajta akadályt). Vannak még kincsek is, és egyéb mellékes tárgyak, pl. lámpa vagy fáklya, amelyek megvilágítják az utat a sötét alagútban. Arra is céloztunk, hogy a lények többek az akadálynál — teljes értékű tárgyak.

Egy kalandprogramban két dolog határoz meg egy tárgyat: az adott hely (a megadott helyiségben) és a bizonyos körülmények között kifejtett hasznosság.

A második tényező teljesen tetszőleges és a programozóra van bízva. Mikor a programozó pl, a kapukat kezelő programrészt írja, tudja, hogy ki kell neveznie egy tárgyat kulcsnak. Választhatja pl. a 11-es tárgyat kulcsnak. Mindössze az teszi kulccsá, ahogyan a kapunyitó programrész működik. A programozó úgy írja meg, hogy ellenőrzi a 11-es tárgy meglétét, mielőtt a kaput kinyithatná. Így van ez más tárgyak esetében is. A 4-es tárgyat csak az teszi kinccsé, hogy a pontozó programrész az ilyen számú tárgyat keresi és sok pontot ad a megtalálásáért. A tárgyak hasznossága rugalmas.

Egy tárgy helye könnyen kézbentartható. A kalandprogramban a szerző létrehoz egy indexes változót, és a helyszín minden tárgyához egy változót rendel hozzá. Ezután egyszerűen beállítja az egyes értékeket annak a helyiségnek a számára, ahol a tárgy található. Ha a 10-es tárgy a 6-os helyiségben van, akkor a tárgyhoz tartozó indexes változóba 6 kerül. Pl. ha a 3-as tárgyhoz tartozó változó értéke 18, ez azt jelenti, hogy a 3-as tárgy a 18-as helyiség padlóján fekszik. Pofonegyszerű!

Egy adott helyszín tárgyainak többsége azért van, hogy a játékos megtalálja és használja. Más szóval, minden tárgy kezdetben egy adott helyen van, egy helyiségben. Mikor a játékos belép a helyiségbe, a program közli vele, hogy a tárgy ott hever. Dönthet úgy is, hogy otthagyja, vagy használhatja a „VEDD FEL" parancsot, hogy felvegye a tárgyat és betegye képzeletbeli hátizsákjába. Ily módon elhordhatja a kincseket a barlangból a támaszpontjára, ahol jutalompontokat kap érte. (Általában van egy előre beállított határa annak, hogy hány tárgyat vihet egyszerre a zsákban.)

Ez módot ad egy találó kérdésre. Mikor a tárgy, mondjuk egy kulcs, a földön van, akkor a helyiségben van. Hol van azonban a kulcs akkor, amikor a játékos viszi? Más szóval, milyen értéket teszünk a tárgyhoz tartozó indexes változóba? Ezt a kétértelműséget úgy oldjuk fel, hogy egy fiktív helyiségszámot rendelünk a hátizsákhoz. Azaz, a tárgyakat akkor hordozzák, mikor a helyet jelző szám megegyezik a zsákéval.

Ha a játékos elejt egy tárgyat, a tárgy helyét jelző szám azonnal átváltozik a jelenlegi helyiség számára.

A *Kardhalak és kincsek*-ben 20 helyiség van. Itt a fel nem használt 21-et rendeljük hozzá a játékos hátizsákjához! (Emlékezzünk arra, hogy a fel nem használt 22 és 23 az erőszakos halálhoz tartozik, amelyet a veszélyes irányok okoznak. A zsák száma hasznos dolog. Ha a játékos leltárt akar készíteni arról, hogy miket hordoz, a program egyszerűen átnézi a tárgyak helyét tároló vektort, és kikeresi azokat a tárgyakat, amelyek helyiségszáma 21. Ezeket a tárgyakat kilistázza.

A helyszínnel kapcsolatos hasznos változók

Egy, a *Kardhalak és kincsek*-hez hasonló kalandprogram sok változót és vektort használ, hogy nyomonkövesse a dolgokat. Épp most beszéltünk meg egy ilyen vektort: a tárgyak helyét tároló vektort. Erre a tárgyak állapotvektora néven hivatkozunk. 16 eleme van, mivel a programban 16 különböző tárgy van. Mikor a program leír egy helyiséget, mindig végig kell néznie a tárgyak állapotvektorát, hogy megállapítsa, vannak-e leírandó tárgyak a helyiségben.

Bizonyos értelemben a játékos maga is tárgy, amennyiben jelen lehet egy bizonyos helyen, és helyiségről helyiségre vándorol. Fenn kell tartani egy változót, az előző vektortól függetlenül, hogy nyomonkövessük, hányas számú helyiségben van most a játékos. Nevezzük ezt a játékos helyét jelölő változónak. Ez a változó elsősorban azáltal változik, hogy a játékos mozgási parancsokat ad ki.

Van még egy tucatnyi hasznos változó, amit időről időre módosítani kell. Szükség van egy számlálóra, amely nyilvántartja a kalandozó megtett lépéseit, mivel ez beleszámíthat az értékelés módjába. Kell egy változó, amely megmondja, hány tárgyat hordoz a játékos, mert a program nem engedélyezi, hogy több tárgyat vegyen föl, mint amennyit elbír. Egy másik változó azt tartja nyilván — szintén értékelési célból —, hogy a játékost hányszor ölték meg. Van még egy pár kevésbé fontos változó, ezeket a későbbiekben érintjük.

Már hosszasan beszéltünk egy másik fontos vektorról — az akadálylistáról. Ennek a tömbnek a mérete a helyszín térképétől és a létrehozott akadályok típusától függ. Erre mindannyiszor hivatkozás történik, amikor a játékos lépni próbál.

Itt kell még ismertetni egy utolsó vektort. Amikor a játékos először belép egy helyiségbe, a program közöl egy-egy bekezdésnyi helyiségleírást, hogy útbaigazítsa a kalandozót. Az első látogatás után a játékost már nem szabad zavarni egy hosszú, részletes leírással. Jobb, ha a program közli, hogy egyszer már volt ott, és csak egy igen rövid leírást ad, egyszerűen közli a helyiség nevét. (Persze, ha ismét igényli a hosszú leírást, akkor kérheti.)

VÁLTOZÓ	JELENTÉS
CT(0)	A JÁTÉKOS HELYE
CT(1)	A MEGTETT LÉPÉSEK SZÁMA
CT(2)	A HORDOZOTT TÁRGYAK SZÁMA
CT(3)	HÁNYSZOR HALT MEG A JÁTÉKOS
CT(4)	A MEGÖLT KARDHALAK SZÁMA x25 PONTTAL
CT(5)	A FELBONTANDÓ SZÁM
CT(6) - CT(11)	AZ ÖSSZERAKANDÓ SZÁM
CT(12)	A KARDHAL MEGJELENÉSÉT VEZÉRLŐ SZÁMLÁLÓ
TX$(0) - TX$(1)	HELYISÉGLEIRÁSOK (MUNKAVÁLTOZÓ)
TX$(2) - TX$(3)	BEOLVASOTT SZAVAK (MUNKAVÁLTOZÓ)
DA(1)	A BEJÁRÁSI TÁBLÁZAT KEZDŐCIME
DA(2)	A SZÓTÁBLA KEZDŐCIME
DA(3)	AZ ÜZENETEK BLOKKJÁNAK KEZDŐCIME
DA(4)	A TÁRGYLEIRÁSOK KEZDŐCIME
DA(5)	A HELYISÉGLEIRÁSOK KEZDŐCIME
RM(1) - RM(20)	A HELYISÉGEK ÁLLAPOTVEKTORA
OB(1,0)-OB(16,0)	A TÁRGYAK ÁLLAPOTA
OB(1,1)-OB(16,1)	A TÁRGYAK HELYE
BK(1) - BK(10)	AZ AKADÁLYLISTA

2-11. ábra. A KARDHALAK ÉS KINCSEK-ben használt indexes változók

Ez a szolgáltatás csak akkor működik, ha van egy jelző, amelyet minden egyes helyiséghez fenntartunk, és ez jelzi, hogy járt-e már játékos a helyiségben. A változó értéke nulla, míg a helyiségbe a játékos be nem lép, azután értéke egyszer s mindenkor egy lesz. Ez a változóvektor a helyiség állapotvektora. A vektor szintén hasznos az értékeléskor, mert rendszerint pont jár a felfedezett helyiség száma után.

Ez az egyszerű igen–nem jelző egy számjegyet foglal el a vektorelemként tárolt egész számból, a többi számjegy a helyiség egyéb jellemzőinek ábrázolására használható fel. Ismét az egész számok felbontása teszi lehetővé ezt a fajta kifejtést.

Összegezzük, mivel is rendelkezünk! A 2-11. ábrán látható a *Kardhalak és kincsek*-ben használt összes változó és vektor, mindegyik rövid magyarázattal ellátva. Néhányat a kevésbé fontos változók közül részletesen ismertetünk a soron következő fejezetekben.

Helyiségek, színhelyek leírása

Eddig túl általánosan, szerkezeti kérdések mélységéig szóltunk a kaland színhelyéről, hogy elmondhassuk az elképzelés mögött rejtőző módszereket vagy annak logikáját. Mikor a játékos a helyszínen megjelenik, ott nem összekötött körök hálóját, se nem táblázatok listáját látja. A színhelyről alkotott benyomását teljes egészében a számítógép képernyőjén megjelenő leírások hozzák létre és tartják fenn. Ezek a leírások „hozzák létre" azt a képet, amelyben a játékos mozog és amellyel érintkezik; az összes vektor és táblázat csak arra való, hogy a megfelelő leírást hívja elő.

Mikor a programozó megalkotja a helyszínt, egy központi elképzelésből indul ki és e köré épít. Javasoltam már jó néhányat — barlangokat, kísértetjárta kastélyokat, irodaházakat. Ha a programozó már elkészült az átfogó színhellyel, a helyiségek ebből már következnek és könnyen adódnak. A programozó egy ívet készít és beszámozza egytől a legnagyobb helyiségszámig (a *Kardhalak és kincsek*-ben ez 1-től 20-ig tartó számozást jelent). Ezt követően a számokhoz felsorolja az összes alárendelt helyet, amellyel a játékos találkozhat ezen a fajta helyszínen.

Tegyük fel, hogy a helyszín egy nagy irodaház! Felsorolhatjuk pl. a titkárságot, a főnök irodáját, az önkiszolgáló éttermet, a hidegvíz-tartályt, a mosdókat. Van aztán még több terem, a folyosó, több iroda, amelyet színük és méretük különböztet meg, esetleg egy másolóhelyiség. Ha már egyszer nekilendült a fantáziánk, valószínűleg sokkal több helyet találunk ki annál, mint amit a maximális helyiségszám lehetővé tesz!

Ne feledkezzünk meg arról sem, hogy az 1-es helyiségszámot rendeljük a támaszponthoz, a tágabb helyszín bejárati pontjához. Példánkban az 1-es helyiség valószínűleg a folyosó lehet vagy az épületen kívüli járda, esetleg az autóparkoló.

Hogy könnyebben hivatkozhassunk rá, a 2-12. ábrán közöljük a *Kardhalak és kincsek* helyiséglistáját. Miközben a helyiségeken gondolkodunk, eszünkbe juthatnak apró megkülönböztető jegyek, amelyeket a későbbiekben beépíthetünk a helyiségek leírásába. Ezeket feljegyezhetjük a rövid név mellé, ahogy a példában látható. Az alapötlet az, hogy minden helyiségnek legyen egy- vagy

#	HELYISÉG	#	HELYISÉG	#	HELYISÉG
1	A GÖDÖR ALJA	9	A UTVESZTŐ	17	ROHANÓ FOLYÓ
2	ROMOK	10	KESKENY SZIKLAPÁRKÁNY	18	NYIRKOS BARLANG
3	FEGYVERTÁR	11	CELLA	19	GÖZÖLGŐ BARLANG
4	VESZTES CSATA	12	IRODA	20	TÜZES CSUCS
5	KRIPTA	13	EBÉDLŐ	"ÁL" HELYISÉGSZÁMOK	
6	JÓSDA	14	PÓKHÁLÓS HELYISÉG	21	HÁTIZSÁK
7	KINCSTÁR	15	B UTVESZTŐ	22	TÜZHALÁL
8	ŐRHELY	16	C UTVESZTŐ	23	LEZUHANÁSOS HALÁL

2-12. ábra. **A KARDHALAK ÉS KINCSEK** helyisegei

kétszavas neve. Ezt használjuk fel a leírás rövid formájaként, amit az egyszer már bejárt helyiségeknél íratunk ki.

A kalandprogramban minden egyes helyiséghez tartozik egy bekezdésnyi leírás, amit DATA utasításban tárolunk. A rövid nevet a hosszú leírással együtt tároljuk. Mikor a játékos bizonyos helyiségbe belép, a program kikeresi a hozzá tartozó DATA utasítást és előveszi ezt a két adatot. Ezután attól függően, hogy a játékos járt-e már a helyiségben, a program vagy a rövid, vagy a hosszú leírást írja ki. Van egy MUTASD parancs is, amely kifejezetten kéri a hosszú leírás ismételt kiírását. Miután a helyiséglistát elkészítettük, a következő próba az egyes helyiségek bővebb leírásának elkészítése.

Öt szabály vagy *irányelv* vonatkozik a helyiségleírások készítésére. Az *első* a különleges forma. A bekezdés teljes szövegének el kell férnie egy BASIC programsorban, megosztva a sort a BASIC DATA utasítással, a rövid névvel és egy elválasztóvesszővel. Ez kb. 240 karakterre korlátozza a bekezdés hosszát, vagy kicsivel több mint három és fél sornyi szövegre a számítógép képernyőjén. Ha a bekezdés vesszőt vagy pontosvesszőt tartalmaz, akkor még az egész bekezdést idézőjelekbe kell tenni, nehogy a BASIC hibásan értelmezze a DATA utasítást. A bekezdésben lehetnek még újsor-karakterek is (ezt a programozó a SHIFT és a „le" nyíl leütésével szúrja be*), ezek javítják a bekezdés külalakját és elkerülhetővé teszik, hogy egy szó két sor közt megtörjön.

A *második irányelv*, hogy adjunk támpontokat az útra vonatkozóan. Csak úgy tisztességes, ha a leírás legtöbb esetben nyíltan megmondja, hogy „ÉSZAKRA EGY KAPU TALÁLHATÓ, ÉS EGY CSARNOK VEZET KELET FELÉ". Ha nem adnánk ilyen támpontokat, a játékost arra kényszerítenénk, hogy végigpróbálja mind a tíz lehetséges irányt, hogy kiutat találjon. Megjegyezzük viszont, hogy nem kell nyíltan megmondani az összes kijáratot; egy esetleges helyiségleírás mondhatja akár azt is: „TÖBB KAPU VAN ITT KÖRÖSKÖRÜL". Emlékezzünk az álcázott utak fogalmára is. Ha az északi fal csak káprázat és valójában kijárat, csak annyit mondjunk: „AZ ÉSZAKI FAL KÜLÖNÖS FÉNNYEL CSILLOG", és bízzuk a játékosra, hogy maga kísérletezzen.

A *harmadik irányelv*, hogy iránytól független nyelvet használjunk. Ezen azt értjük, hogy a programozó ne tételezzen fel semmit arról, hogyan lépett be a játékos a helyiségbe, amit éppen vizsgál! Képzeljünk el pl. egy helyiséget, amelynek két bejárata van; fenn egy csapdaajtó és egy acélrács a falon! Ostobaság kiírni, hogy „EGY NYÁLKÁS HELYISÉGBE ZUHAN". Még abban az esetben is, ha a csapdaajtó lenne a helyiség elsődleges bejárata, akkor is ott van a rács. Mi történik, ha valamikor később a játékos ismét belép a helyiségbe a rács felől? A leírás pontatlan lenne. Mindig úgy írjuk le a helyiséget, mintha

* A HT-108Z gépen csak „le" nyíl.

a játékos hirtelen és váratlanul jelent volna meg a kellős közepén! Írjuk le a legtöbb bejáratot és kijáratot, azt is, amelyen át nagy valószínűséggel éppen beverekedte magát!

A *negyedik irányelv* az, hogy kerüljük nem létező tárgyak használatát; más szóval olyan tárgyakét, amelyet maga a program nem támogat. Ez nehéz feladat, néha egyenesen kivihetetlen, de próbáljuk betartani! Elkerülhetetlen bajt okoz, ha olyan tárgyat tüntetünk fel a helyszín részeként, ami nem szerepel a tárgyak állapotlistáján. Miért? Azért, mert ha a leírás azt állítja, hogy ott van, a játékos okvetlenül megpróbálja fölvenni! Ha a program nem számol létezésével, ez az álobjektum felborítja a szimuláció élethűségét azáltal, hogy nem reagál semmilyen parancsra, sőt esetleg a programfutást is megszakíthatja. Ha kénytelenek vagyunk bevenni nem programozott tárgyakat a leírásba, tegyünk hozzá valamit, amely elveszi a játékos kedvét attól, hogy megpróbálja elmozdítani! Ha a színhelyben szerepel egy hidegvíz-tartály, mondjuk azt, hogy „A HIDEGVÍZ-TARTÁLY A KÖZELBEN VAN, A FALHOZ ERŐSÍTVE". Legalább van valami magyarázat arra, ha gyengécske is, hogy miért nem akar egy tárgy tárgyként viselkedni.

Az ötödik és egyben utolsó *irányelv* az, hogy a leírásnak meg kell ragadnia a képzeletet! A kalandprogram realizmusa nem kevéssé múlik azon, hogy a történet előadása színpompás, talányos legyen. Sok módja van annak, hogy a leírás „belopja magát" a játékos agyába. Az érdekes színek, furcsa, meglepő méretek és formák mind izgalmasabbá, érdekfeszítőbbé tehetik a helyszínt. Milyen vajon a helyiség? Hideg, nyirkos vagy forró, száraz, esetleg sötét, homályos, vagy pedig tengeri hínártól bűzlő? A lényeg az, hogy a kimondott szavakon túl képzeteket közvetítsünk.

A kalandok helyszínének leírásakor kedvenc eszközünk lehet a valószerűtlenség. Használjunk néhány olyan helyiséget, amely látszólag egyáltalán nem illik a napszakhoz vagy a helyszín hangulatához! (Ez vonatkozik a tárgyakra és a lényekre is!) Pl. a *Kardhalak és kincsek* egy föld alatti manókirályságot ír le barlangokkal, jósdával, fegyverteremmel, de „bevetünk" egy irodát és egy ebédlőt is, épp csak a meglepetés kedvéért. Szinte minden megengedett, ha a játékos érdeklődésének felkeltéséről van szó! Miért ne lehetne a *Marsbeli városban* egy nagy vörös épület, benne egy röpködő tűzoltófecskendővel? Miért ne lehetne a víz alatti *Atlantiszon* egy zuhanyfülke, amely levegőt spriccel? Vagy pl, miért ne lehetne a *vadnyugati* helyszínen egy sarki lóetető „normál" és „szuper" zabot adagoló kutakkal? Az Olvasó már biztosan érti, miről van szó!

A helyiségek leírására vonatkozó utolsó megjegyzés az útvesztőkhöz kapcsolódik. Emlékezzünk rá, hogy egy útvesztő összes helyiségéhez azonos leírás tartozik, hogy megzavarjuk a vándorló játékost! Ez rendszerint ilyen jellegű: „ELTÉVEDT EGY ÚTVESZTŐBEN" vagy „EZ EGY JELLEGTELEN HELYISÉG".

Az útvesztő helyiségeire vonatkozó rövid, ill. hosszú leírás szükségszerűen megegyezik. Hogy miért? Hát azért, hogy a játékost megtévesszük: ne tudja,

hol is van tulajdonképpen. Ha valamelyik helyiségről rövidebb leírást kap, tudni fogja, hogy már járt ott. Véssük emlékezetünkbe: amikor a DATA utasítás megírására kerül sor, a rövid és a hosszú leírás legyen azonos, akkor is, ha a helyiséglistában más a nevük — A útvesztő, B útvesztő, C útvesztő.

Hogyan írjuk le, hogy mit találunk?

Ha már összeállítottuk a helyiségek listáját, és megírtuk a helyiségek leírását, ideje elkészítenünk a tárgyak listáját és a hozzá tartozó leírást. Amikor a játékos belép egy bizonyos helyiségbe, megkapja a helyiségleírást. Ha tárgyak is vannak körülötte, mindegyikről kap egy-egy sornyi leírást. Az egysoros leírás a tárgy bővebb leírása. Mindegyik tárgynak van egy rövidített leírása (a helyiségekhez hasonlóan), ez egy- vagy kétszavas név. Ezt a nevet akkor használjuk, ha a játékos a LELTÁR parancsot írja be, mert meg akarja vizsgálni a zsák tartalmát.

A 2-13. ábrán látható a *Kardhalak és kincsek*-ben használt tárgyak listája. A lista három fő részre bomlik, ezekkel egyenként és részletesen foglalkozunk.

Először a kincs jellegű tárgyakat kell létrehozni. Ezek az értékes tárgyak, megtalálásuk a kaland elsődleges célja. Képzeletbeli kémkalandokban ezek a kincsek pl. bizalmas állami iratok, amelyeket el kell lopni. A *Kardhalak és kincsek*-ben ezeknek a tárgyaknak egyszerűen pénzben kifejezhető értéke van, mint pl. amilyen a törpék kincseskamrájában található.

Egyszerűsíti a dolgokat, ha a kincsek a lehető legkülönbözőbbek — legalábbis ami a nevüket illeti. Ennek az a célja, hogy a kalandprogram „ne jöjjön zavarba", mikor megpróbálja meghatározni, melyik kincsre hivatkozik a játékos. Olyan neveket használjunk, amelyek eléggé megkülönböztetnek két kincset! Pl. minden drágakő típusú kincs nevében szerepeljen egy meghatározott drágakőtípus; ne mondhassa a játékos: „VEDD FEL A DRÁGAKÖVET"; pontosítania kell: „VEDD FEL A GYÉMÁNTOT." Még az 1-es tárgy sem egyszerűen drágakő, hanem drágaköves korona.

Emlékezzünk vissza a valószerűtlenség elvére, amely érdekesebbé teszi a dolgot! Teljesen elfogadható és mókás lenne, ha a kincsek között lenne egy törpe méretű tranzisztoros rádió.

A következő tárgycsoport: az eszközök. Ezek olyan tárgyak, amelyekre szükség van azoknak a különféle akadályoknak a legyőzésében, amelyek gátolják a kalandozót a kincsek megszerzésében. Az eszközök erősen függnek az akadályoktól, és helyszínről helyszínre mások és mások. Barlang jellegű helyszíneken, amilyen a *Kardhalak és kincsek* is, szokásos a lámpa vagy fáklya, mivel a föld alatti környezet szükségessé teszi ezt a fajta eszközt. A programot úgy kell alakítani, hogy attól függően korlátozza a játékos föld alatti mozgását, van-e nála lámpa vagy fáklya. Eszköz hiányában a helyiségek leírásának kiírása le van tiltva, és a következő üzenet jelenik meg: „VIGYÁZAT! ITT NAGYON

SÖTÉT VAN!" Természetesen ilyen jellegű eszköznek semmi keresnivalója nincs nyitott helyszínen, napfénynél.

A kulcs szintén elengedhetetlen tartozék a kapu jellegű élettelen akadályok miatt. A program nem hajlandó kinyitni a kaput vagy a rácsot, ha nincs a kulcs a játékos hátizsákjában. Ha a programozó igazán fantáziadús, használhat két- vagy háromfajta kulcsot, amelyek mindegyike egy bizonyos kaput nyit.

A többi eszköz alapvetően fegyver. A program felépítése olyan, hogy bizonyos lényeket csak egyik vagy másik fegyver pusztít el. A helyszíntől függően lehet vadászpuskától a lézerágyúig bármi. Lehet, hogy önmagukban veszélytelennek tűnnek; lehet, hogy pont egy üveg közönséges szódavíz állítja meg a portyázó robotot, úgy, hogy berozsdásítja.

Végül a lények csoportját kel kitalálni. Ezek a rémek állják el a félhomályos barlangok átjáróit; egy barbár színhely vad őslakói, a marslakók vagy a titkosrendőrség. Ne felejtsük, hogy ezek a lények szintén akadályok, és mindegyikhez tartozik egy bejegyzés az akadályok listáján. Mikor valamelyiket képzeletbeli kardpárbajban legyőzzük, az akadályok listáján a bejegyzés megváltozik, és a lényt halottnak tekintjük. (Hogy a dolgokat leegyszerűsítsük, a lény általában eltűnik, nem hagyja hátra a tetemét. Ezt úgy érjük el, hogy a tárgyak állapotlistáján a lény helyzetéhez nullát írunk, ez megfelel a hirtelen eltűnésnek, mivel nullás helyiség nincs.)

Mennyi lehet a háromféle tárgyból? Ezt a tár mérete szabja meg, mivel minden tárgyhoz tartozik egy változó a tárgyak állapotvektorában, egy rövid és egy bővebb leírás, és programkód, amely a hozzá kapcsolódó funkciókat kezeli. A túl sok kincs elveszi a keresés izgalmát; a túl sok lény unalmas. A 2-13. ábrán feltüntetett tárgyak száma és aránya valószínűleg optimális egy 20 helyiségből álló helyszín esetében.

Említettük már, hogy a helyiségekhez hasonlóan a tárgyaknak is van rövidebb és bővebb leírása — a bővebb a leltár listázásához. Milyen irányelveket ajánlhatunk ezekhez?

A rövid leírás a tárgy egy- vagy kétszavas neve — eddig egyszerű. A bővebb leírás hossza valahol 48 és 96 karakter között van, ami legföljebb másfél sornyi szöveget tesz ki a képernyőn. Rendszerint ilyen jellegűek: „EGY ÜRES LAP HEVER A FÖLDÖN." A kincsek leírása rendszerint felkiáltójellel végződik, valahogy így: „ITT ÁLL EGY GYÖNYÖRŰ SZOBOR SMARAGDOKKAL KIRAKVA!" A lények is rendszerint felkiáltásra sarkallnak, és leírásuk még nyomatékosabb, pl. „EGY ÓRIÁS SZŐRÖS MAMMUT VAN A KÖZELBEN, SZINTE MÁR TOMBOL!"

Csak arra kell vigyázni, hogy a leírásban kerüljünk minden utalást a közvetlen környezetre, mivel az megváltozhat! Pl. ne mondjuk, hogy „EGY CSILLOGÓ FÉMPÉNZ VAN A KASSZÁBAN", mivel a játékos esetleg elveszi, és egy olyan helyiségben ejti le, ahol nincs is kassza. Ettől a szabálytól kissé

eltérhetünk lényeknél, mivel azok rendszerint egy helyiségben élik le életüket. (Segítségünkre van, hogy a program egyik része megakadályozza, hogy a játékos felvegye a tűzokádó sárkányt és „elfuvarozza" a hátizsákjában.)

Ha elkészítettük a tárgyak listáját és befejeztük a leírásokat, olyan jelzésekkel kell kiegészíteni a listát, amelyek megmondják, melyik helyiségben vannak a tárgyak a játék kezdetekor. Figyeljük meg ezt az információs hasábot a 2-13. ábrán a tárgyak neve mellett! A programozón áll, hogy megválassza ezt a helyet, de adunk néhány ötletszerű tanácsot! Az eszközöket tegyük oda, ahol kissé lelassítják az előrehaladást. Más szóval, ha a helyszínen van egy kulcs, ne adjuk azonnal a játékos kezébe; tegyük a színhely mélyére, így visszafelé is végig kell járnia az utat, hogy használhassa. A kincseket helyezzük lezárt kapuk és dühös lények mögé, de néhányat hagyjunk elöl, őrizet nélkül — étvágygerjesztőként!

A helyiségszámoknak ez a sora, a tárgyak kiindulási helye, később DATA utasítás formájában kerül a RAM memóriába. A program indulásakor egy inicializáló programrész egyszerűen beolvassa ezeket a számokat, és az addig kitöltetlen tárgyak állapotvektorába tölti be.

A váratlan ellenfél

Mielőtt pontot tennék ennek a fejezetnek a végére, amely a színhely létrehozásáról szól, hátra van még valami, amit a programban meg kell valósítani. Magyarázatként vegyük figyelembe, hogy az eddig említett lények igencsak szelídek és meglehetősen jól kezelhetőek!

Meg kell adni, épp elég vérszomjasak, ha megtámadják őket, de épp ez az — passzívak. A kalandozó félelem nélkül sétálhat abban a helyiségben, ahol az óriás ájtatos manó van, mindaddig, míg meg nem támadja a fenevadat. Hát miféle kihívás ez!

Szükség van még valamire, nevezzük „őt" *kitartó lénynek!* Kitartásából következik, hogy nem hajlandó magára hagyni a kalandozót. Az ilyen ellenfelet három dolog jellemzi: szabadon kóborol a helyszínen, provokáció nélkül támad és helyiségről helyiségre követi a játékost.

Láthatjuk, micsoda hatalmas ellenfél az ilyen lény. Összevissza kóborol, míg ugyanabban a helyiségben bukkan fel, ahol a kalandozó. És támad! A játékos megpróbál menekülni, de legnagyobb rémületére az ocsmány lény vele tart! Le kell őt győzni, vagy ő fal föl.

Világos, hogy a kitartó lény teljesen más, mint a többi passzív lény és teljesen másképp kell kezelni! (Érdemes tanulmányozni a 4. fejezetben a félelmetes Kardhal megalkotását.)

OBJEKTUM	TIPUS	LEIRÁS	A KIINDULÁSI HELYISÉG SZÁMA
1	KINCSEK	ÉKKÖVES KORONA	4
2		ARANYKOCKA	7
3		GYÉMÁNTBOGÁR	20
4		EZÜSTŐV	11
5		PLATINAGYŰRŰ	5
6		CSISZOLT ÓNIX	19
7		MILLIÓKAT ÉRŐ ÉRME	7
8		HOMOKÓRA	6
9	ESZKÖZÖK	FÁKLYA	2
10		BŰVÖS SZEKERCE	3
11		KULCS	10
12		BŰVÖS GRÁNÁT	12
13	LÉNYEK	ÓRIÁS ÁJTATOS MANÓ	4
14		ÓRIÁSGYIK	18
15		FEHÉR PÓK	14
16		NEVESINCS BORZALOM	6

2-13. ábra. A KARDHALAK ÉS KINCSEK objektumlistája

A majdnem teljesen befejezett helyszín

Amint az Olvasó elolvasta ezt a bekezdést, gratuláljon magának, milyen messze eljutott (feltéve, hogy nem hagyott ki oldalakat)! Ez a fejezet tartalmazta formai szempontból a kalandprogram készítésének lényegét. A könyv hátralevő részében azzal foglalkozunk, hogy sorravesszük a fejezetben megismert fogalmakat és BASIC-ben kódoljuk őket. Abban az esetben, ha az Olvasó végig

verejtékezett és azon törte a fejét, hogyan képzelhette, hogy boldogul ezzel a fajta programozással, nyugodjon meg — a kemény alapozómunka befejeződött!

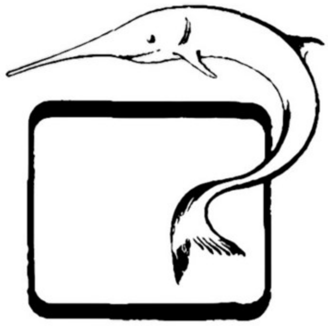
Röviden tekintsük át a kaland helyszínének alapelemeit! Amíg egyesével felsoroljuk őket, ellenőrizze Ön is, sikerül-e visszaemlékeznie a rendeltetésükre és a funkciójukra! Ha bizonytalan néhányban, lapozzon vissza, és alaposan ismételje át!

1. A helyszín helyiségekből áll.
- Szükség van a helyiségek listájára a helyiségek rövid nevével.
- Kell egy bővebb leírás minden helyiségről.
- Meg kell adni a helyiségek állapotvektorát, ez jelzi, hogy még nem voltunk egy helyiségben.
- Szükség van a helyszín térképére a helyiségek egymás közti kapcsolatával.
- Kell egy bejárási térkép, amely megadja a bejáratokat és a kijáratokat.

2. A helyszínt akadályok nehezítik.
- Szükség van éiő akadályokra; ilyenek a lények.
- Szükség van élettelen akadályokra, mint pl. lezárt kapuk.
- Kell egy akadálylista, ez tartalmazza a leküzdendő akadályokat.

3. A helyszínt tárgyak töltik meg.
- Szükség van kincsekre, eszközökre és lényekre.
- Szükség van a tárgyak rövid nevét tartalmazó tárgylistára.
- Szükség van minden tárgyról egy bővebb leírásra.
- Szükség van a tárgyak megtalálásához a tárgyak állapotlistájára.

Most végre van elképzelésünk szinte mindarról, ami egy kalandprogram működéséhez kell! Haladjunk tovább a következő fejezetre, és lássuk, hogyan használhatjuk fel az alapokat BASIC programban!



3. FEJEZET

A program strukturálása

Sajnos, a legtöbb BASIC program tervszerűtlenül készül. A programozó egy apró ötletből indul ki, és megír egy egyszerű programot. Később új szolgáltatásokkal bővíti programját, a kód gyorsan és egyenetlenül növekszik. Végre elkészül egy masszív, ormótlan termék. Csodák csodája a program fut, de ha szükség van itt egy kis javításra, ott egy kis módosításra, akkor a programozó elakad. Hol is van a nyomtatót meghajtó programrész? Hol van a változókat frissítő programrész? Elveszve a burjánzó programban, a programozó azt sem tudja, mit is keresgél!

Aki kalandprogramot ír, legalább három oka van rá, hogy ne legyen pongyola. Első a rendelkezésre álló tár. Egy *Kardhalak és kincsek*-szerű programnak szüksége van minden egyes föllelhető byte-ra. A pongyola kód valószínűleg redundanciákat és más, rossz hatásfokú elemeket tartalmaz, amit jobb szubrutinba szervezni. Egy másik tényező a sebesség. A kalandprogram a lehető legtöbb munkát próbálja elvégezni a lehető legrövidebb idő alatt. A pongyola kódot igen nehéz áramvonalasítani. Az utolsó tényező a módosíthatóság. Aki elég vállalkozó kedvű ahhoz, hogy kalandprogramot írjon, előbb-utóbb valami módon fejleszteni akarja — további helyiségekkel, új lényekkel, több kinccsel. A pongyola kód javítása annyi kudarccal jár, hogy meg sem éri.

Az előbbi okok miatt a kezdet kezdetétől fegyelmezett, körültekintő módon írjuk a programot! Tartsuk magunkat a strukturált programozás szabályaihoz és élvezzük annak előnyeit!

Strukturálás fájdalommentesen

Semmi sem hangzik olyan baljóslatúan, mint a strukturált programozás. Véleményem szerint homályos jelölésmódban készített terjedelmes diagramok képzetét kelti, több ívnyi folyamatábráét, és más hasonlókat. Menekülésre készteti a jövő kalandprogramíróit.

Igazában, ha belegondolunk, a program struktúrájának csírája benne rejtőzik magában a BASIC-ben, ránk mered a képernyőről. Ez pedig a sorszám. Gondolkodjunk el egy pillanatra! A Microsoft BASIC-ben a programozó 0-tól 65529-ig bezárólag tetszőleges sorszámot használhat. Általában egyszerűen megszámozza a sorokat: 10, 20, 30 és így tovább. Azután sorokat présel közéjük, ha később újabb szolgáltatásokkal kell bővítenie programját. Mindig csak parányi részét használja a rendelkezésére álló számoknak.

Mit jelent ez? Mindössze ennyit: ha olyan sok szám áll rendelkezésre, hogy lehetőségünk van véletlenszerűen választani, akkor értelmesen válasszuk ki őket! Más szóval, bizonyos feladatokhoz *hozzárendelhetjük* a sorszámok bizonyos halmazát. Ezután, ha valaha változtatni kell, tudjuk, hol kell a listán keresni. Ahelyett, hogy a fejünket vakarnánk és azt mormolnánk: „Hm, úgy emlékszem, az a programrészlet a 639-es vagy a 369-es, vagy valamelyik hasonló sorban van", tudjuk, hogy „Ez egy inicializáló programrészlet; valahol a 0-s és a 99-es sor között van".

A strukturált programozás első kulcsa, hogy szolgálatunkba állítsuk a sorszámokat. Úgy használjuk, mint a fenti példában, hogy könnyen olvashatóvá tegyük a programot. Később látni fogjuk, hogyan tehető segítségével gyorsabbá a DATA utasításokra történő hivatkozás is.

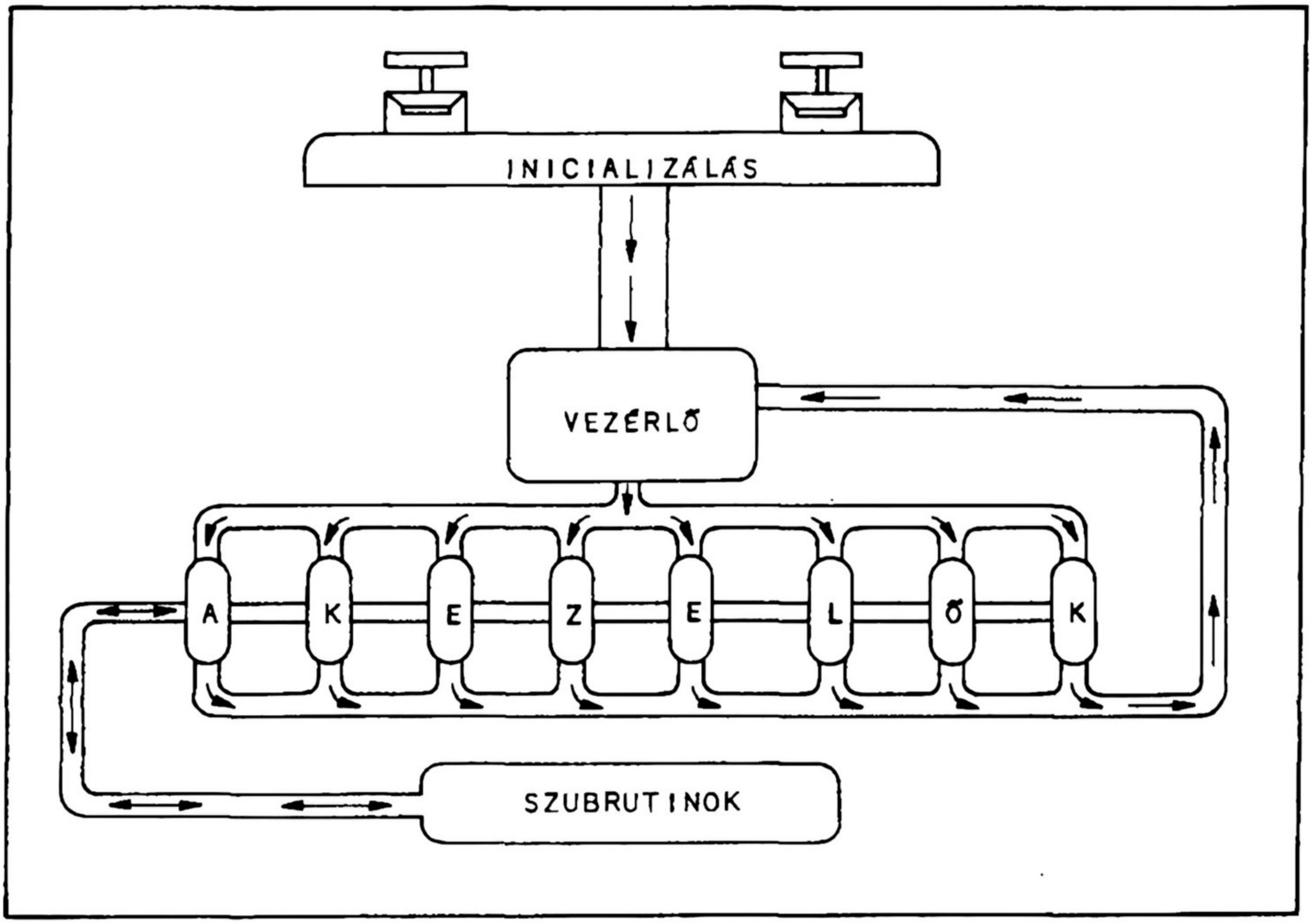
A strukturált programozás második kulcsa a program vezérlési struktúrájával kapcsolatos. Ez azt jelenti, hogy előre át kell gondolni, hogyan is végzi el a program feladatát. Ha jobban megnézzük a programvégrehajtás folyamatát, látni fogjuk, hogy sok olyan feladat van, aminek hasonló a kezelése, sok az ismétlődő út. Ha már tudjuk, melyek ezek, úgy írhatjuk meg a programot, hogy a kód bizonyos részei a program több funkcióját is ellátják, nemcsak egyet. Ez hatékony és tömör programot eredményez — pont ilyenre van szükség, ha csak 16 K tár áll rendelkezésre! Két olyan vonása van a Microsoft BASIC-nek, amely lehetővé teszi ezt a fajta áramvonalasítást. Egyik a GOSUB, RETURN pár, amely lehetővé teszi a szubrutinhívást. A másik az ON X GOTO lehetőség, az a képesség, hogy kezelőkre ugorjunk. Ezeket a módszereket teljesen kihasználjuk a *Kardhalak és kincsek*-ben.

Mielőtt túl mélyre tévednénk a programozás általános kérdéseiben, térjünk vissza kalandprogramunkra!

Az áramlást figyelve

Vannak olyan szennyvíztisztító reklámok, amelyekben átlátszó szennyvízvezetékeket látni. Ezekben könnyen követhető a víz áramlása, mert a csövek átlátszóak. Nem lenne jó, ha a programozás is hasonló lenne? Meg lehetne állapítani, a program mely részeire esik a legnagyobb terhelés.

A 3-1. ábra nem más, mint egy kalandprogram áramlásának szemléltetése átlátszó vezetékekkel. Ez a fajta diagram abban segít leginkább, hogy a konst-



3-1. ábra. A program vezérlési szerkezetének szemléltetése átlátszó vezetékekkel

rukció egyszerűsítése érdekében logikai részekre osszuk a programot. Nézzük egyesével a csöveket és szerepüket!

A program első része az *inicializáló programrész*. A kód itt csak egyszer hajtódik végre, és az a feladata, hogy előre meghatározott kiindulási állapotba hozza a helyszínt. Milyen jellegű dolgokat foglal magában az inicializálási eljárás? Egyrészt az összes megfelelő méretű numerikus változó tömböt kell létrehozni. Ezt követően be kell állítani a változókat, hogy helyesen szimulálják a helyszínt. Pl. a játékos helyét jelző változót a támaszpont számára kell beállítani.

A folyamat egyszerűsítésének érdekében a kódnak sok más része is az inicializáló programrész rendelkezésére áll. Ezek DATA utasítások, amelyek a változók és tömbök beállításához szükséges értékeket tartalmazzák. Ezek egyike az *akadályokat inicializáló blokk*. Az inicializáló kód kiolvassa az értékeket ebből a blokkból és betölti az akadályok listája néven ismert tömbbe (emlékezzünk vissza, ez foglalkozott a kapubejáratokkal és a lényekkel). Egy másik a *tárgyakat inicializáló blokk*. Ez tartalmazza a helyszín összes tárgyának kiindulási helyét, ezeket a tárgyak állapotmátrixába töltjük. Harmadik blokk a *helyiségeket inicializáló blokk*. Ennek segítségével töltjük fel a helyiségek

állapotvektorát, amely azt jelzi, mely helyiségekben jártunk már, melyekben nem.

Egy kérdés biztosan felvetődik: miért van szükség egy hosszú DATA listára a helyiségek állapotvektorának feltöltéséhez? Nem úgy indul-e az összes helyiség, hogy ott még nem jártak? Nem felel meg egy egyszerű programciklus, hogy segítségével nullát töltsünk a helyiségek állapotvektorába? Erre a kérdésre a válasz a későbbi bővítés. Elképzelhető, hogy a későbbiekben más állapotjellemzőket is számon kell tartani a helyiségekkel kapcsolatban. Van értelme annak, hogy lehetőséget biztosítsunk speciális értékek betöltésére, habár a programnak ebben az első változatában minden elem nullával egyenlő. Emlékezzünk vissza arra, hogy a jövőben lehetőség lenne a helyiség álapotát több számjegyre felvontani; az 1. számjegy lehetne a bejárt/be nem járt jelző. A többi számjegy más állapotjelzőket jelölhetne. Egyelőre fogadjuk el, hogy ez hasznosnak bizonyulhat egy későbbi időpontban!

Most, hogy a színhely inicializálása megtörtént, komolyan elkezdődhet a játék. A kód következő részének neve *Vezérlő*. Ezt a nevet onnan kapta, hogy a programnak elsősorban ez a része felelős a játék vezérléséért, az összes többi rész ennek alárendeltje vagy végül ide tér vissza. (Megfigyelhetjük, hogy a programban található sok csővezeték előbb-utóbb visszatér a Vezérlőhöz.) A Vezérlőnek két alárendelt része van. Egyik a *leírórész*. Ez a programrész írja le azt a helyiséget, amelyben a játékos tartózkodik, beleértve a tárgyakat és a közelben található ellenségeket. A másik a *parancsrész*. Ez a programrész fogadja a billentyűkről jövő üzenetet és értelmezi a játékos szándékát.

A leíró alárendelt rész két DATA blokkot használ, és akinek jól fog az esze, meg tudja mondani, melyek ezek! Egyik a *helyiségeket leíró blokk*. Ebben tároljuk az egyes helyiségek rövidebb és bővebb leírását, helyiségenként egy DATA sorban. A másik a *tárgyakat leíró blokk*, amely hasonlóan, tárgyanként egy DATA sorból áll és a tárgy rövidebb és bővebb leírását tartalmazza.

Van még egy DATA blokk, amelyet minden alkalommal használunk, mikor a játékos parancsot ír be. Ez a *szótábla* lényegében a kalandprogram alapszókincsét tartalmazza, megfelelő számokkal együtt, amelyek a beírt szó meghatározásában segítenek. A parancsrész mindig átnézi a szótáblázatot, mikor beírunk egy szót. Ha a szó nem található a táblázatban, az értelmező nem tud értelmesen reagálni, ezért további információt kér a játékostól.

A kód következő szintje a Vezérlő alatti programegységek sora. *Kezelőknek* nevezzük őket. A kalandprogram minden lehetséges funkciójához egy kezelő tartozik. A Vezérlő parancsrésze meghatározza, melyik kezelőt kell kiválasztani és végrehajtani. Pl. ha a játékos azt írja, „LELTÁR", a parancsrész kikeresi a szótáblázatból ezt a szót. Mikor megtalálja, az értelmező kiolvassa a táblázatból azt a számot is, amelyik meghatározza, melyik kezelőt kell végrehajtani. A szinonímák ugyanazt a kezelőt hívják meg. Könnyen átlátható, hogyan növelhetők a program képességei ezzel a módszerrel. A programozó egyszerűen egy új parancs kulcsszavával bővíti a táblázatot és elkészíti

az új funkció kezelőjét. A kód többi része változatlan, de a kalandprogram most már egy új parancsot is felismer. (Ebből is látható, hogyan csökkenti a strukturált programozás a programbővítés fáradságát.)

A mozgás irányításáért felelős kezelők hivatkoznak az előző fejezetben létrehozott nagyon fontos DATA-blokkra. Ez a bejárási táblázat, amely minden egyes helyiséghez megadja a be- és kilépéshez szükséges információt. A mozgást irányító kezelő és a bejárási táblázat valószínűleg a programkód legjobban igénybe vett része, leszámítva a Vezérlő parancsrészét.

A kalandprogram két utolsó része az összes kezelőt kiszolgálja, még a Vezérlőt is. Közülük az első az üzenetek blokkja nevű DATA rész. Sok kezelőnek van szüksége olyan speciális üzenetekre, amelyeket azért kell kiírni, hogy a kezelő állapotát jelezze. Ha falba ütközünk, egy üzenet közli: „ARRA NEM MEHET!" Ha lelépünk egy szikláról, egy üzenet „VESZTÉBE ROHAN..."-nal figyelmeztet. A szó szoros értelmében több tucat hasonló, egysoros üzenetet kell használatra készen tartani.

Az utolsó rész tartalmazza a programban hívott szubrutinokat. Az egyik szubrutin egy DATA blokk bejegyzéseit keresi ki. Egy másik a bejárási táb

A PROGRAM SZERKEZETE

SORSZÁMOK	PROGRAMSZEGMENS
0-99	INICIALIZÁLÁS
100-199	FŐ VEZÉRLŐ
200-999	KEZELŐK
1000-1999	SZUBRUTINOK
2000-2999	HELYISÉGEKET INICIALIZÁLÓ BLOKK
3000-3999	OBJEKTUMOKAT INICIALIZÁLÓ BLOKK
4000-4999	AKADÁLYOKAT INICIALIZÁLÓ BLOKK
5000-5999	BEJÁRÁSI TÁBLÁZAT
6000-6999	SZÓTÁBLÁZAT
7000-7999	ÜZENETEK BLOKKJA
8000-8999	OBJEKTUMOKAT LEIRÓ BLOKK
9000-9999	HELYISÉGEKET LEIRÓ BLOKK

3-2. abra. Melyik programszegmenshez milyen sorszamtartomany tartozik

lázatot elemzi. Van olyan szubrutin is, amely az akadálylistán valamelyik bejegyzés állapotát változtatja meg. Sok más, gyakran használt funkció szubrutinban található, ezek egy nagy közös területen helyezkednek el.

Lépjünk hátra egy lépést, és nézzük meg, hogyan helyezzük el ezt a sok programrészt a BASIC sorszámoknak megfelelően!

A 3-2. ábra egy kalandprogram struktúrájának a listája. Figyeljük meg, hogy az inicializáló kód (amely elsőként hajtódik végre) az első a program struktúrájában! Akárhol lehet a 0 és 99 közötti sorszám tartományában. (Egy blokkon belül a páros sorszámokat fogjuk használni. A *Kardhalak és kincsek* inicializálása a 2, 4, 6, 8 stb. sorszámokon fut.)

A következő a Vezérlő, aztán a kezelők és a szubrutinok. Figyeljük meg, hogy a kezelőknek és a szubrutinoknak több helyet hagytunk, mint a Vezérlőnek, mert valószínű, hogy ez a két rész még bővülni fog. (Természetesen, még így sem fogjuk kihasználni a rendelkezésre álló több száz sorszámot.)

Ezt követik ezresével növelve a DATA blokkok. Elöl áll az a három blokk, amelyiket csak az inicializáló eljárás használ. Utána következik a szótár és a helyiségek útjainak két táblázata. Legvégül pedig a három szövegblokk; az üzeneteké, a helyiségek és a tárgyak leírásáé. Rá fogunk jönni, hogy ezek a szövegblokkok fogyasztják igazán a tárat.

Fogd az adatot, és fuss!

A fenti tizenkét programrészből nyolc adatblokk. Ezek BASIC sorokban elhelyezett hosszú szám- és szólistákból állnak, vesszők választják el őket és a sorok elején a DATA kulcsszó áll.

Lehet, hogy az Olvasó még nem jött rá, de a BASIC DATA-ból nehéz feladat adatokat kiszedni. Két BASIC parancs kapcsolódik ehhez a folyamathoz. Az első a READ. Valahányszor végrehajtódik, a DATA egy eleme bekerül egy kiválasztott változóba. A következő READ utasítás veszi a következő elemet, míg csak mind be nem lett olvasva. A másik a RESTORE parancs, amely elölről kezdi a READ folyamatát a programelső DATA sorának legelső elemétől.

Egy bizonyos pontig ez így mind szép és jó! Mit tegyünk, ha egy 300 elemű DATA lista 173. elemére van szükségünk? BASIC-ben erre egy mód van: hajtsunk végre RESTORE-t, aztán futtassunk 173-szor egy ciklust, amely READ parancsot hajt végre, amíg a 173. elemet olvassuk be!

Tételezzük fel, hogy ezután a 160. elemre van szükség! Lehet-e visszafelé olvasni, hogy hozzáférjünk? Valahogy oda tudnánk-e ugratni ahhoz az elemhez? Nem, elölről kell kezdeni egy RESTORE-ral, aztán READ, READ, READ, míg meg nem találjuk. A BASIC DATA sorokkal az a baj, hogy *soros hozzáférést* tesznek lehetővé, más szóval minden elemet el kell olvasni sorjában, egyet sem lehet kihagyni.

Feltehető a kérdés: „És aztán? Csinálunk egy egyszerű FOR-NEXT ciklust, hogy végrehajtsa az összes haszontalan READ-et." Rendben van; de nézzük a nehézségeket! Első az idő. Kalandprogramunk egy teknősbéka sebességével halad, ha az összes DATA elemet sorosan kell átolvasnia. (Emlékezzünk arra, hogy a program kétharmada adat!) Azután azt is ki kell számítani, hányszor kell a ciklust futtatni! Hány ciklusra van szükség, ha a 4-es DATA blokk 17-es eleme kell? El kell ugye indulni az első blokknál és át kell olvasni, akár szükség van rá, akár nem, hála a RESTORE parancsnak. Hogy végül megkapjuk, 17-et hozzá kell adni a másik három blokk együttes hosszához. Ez már munka a javából!

Még egy nehézség a végére: az egyik blokkban szám jellegű adatok lehetnek, a másikban szövegek. Ha ismételten megpróbálunk READ A utasításokat végrehajtani, más szóval, ha az A numerikus változóba próbáljuk tölteni az adatokat, a program hibára fut, ha a ciklus olyan blokkhoz ér, amelyben betűkből álló szöveg van! Tehát az ember nem lehet válogatós BASIC-ben! Semmilyen körülmények között nem lehet egy blokkot kihagyni.

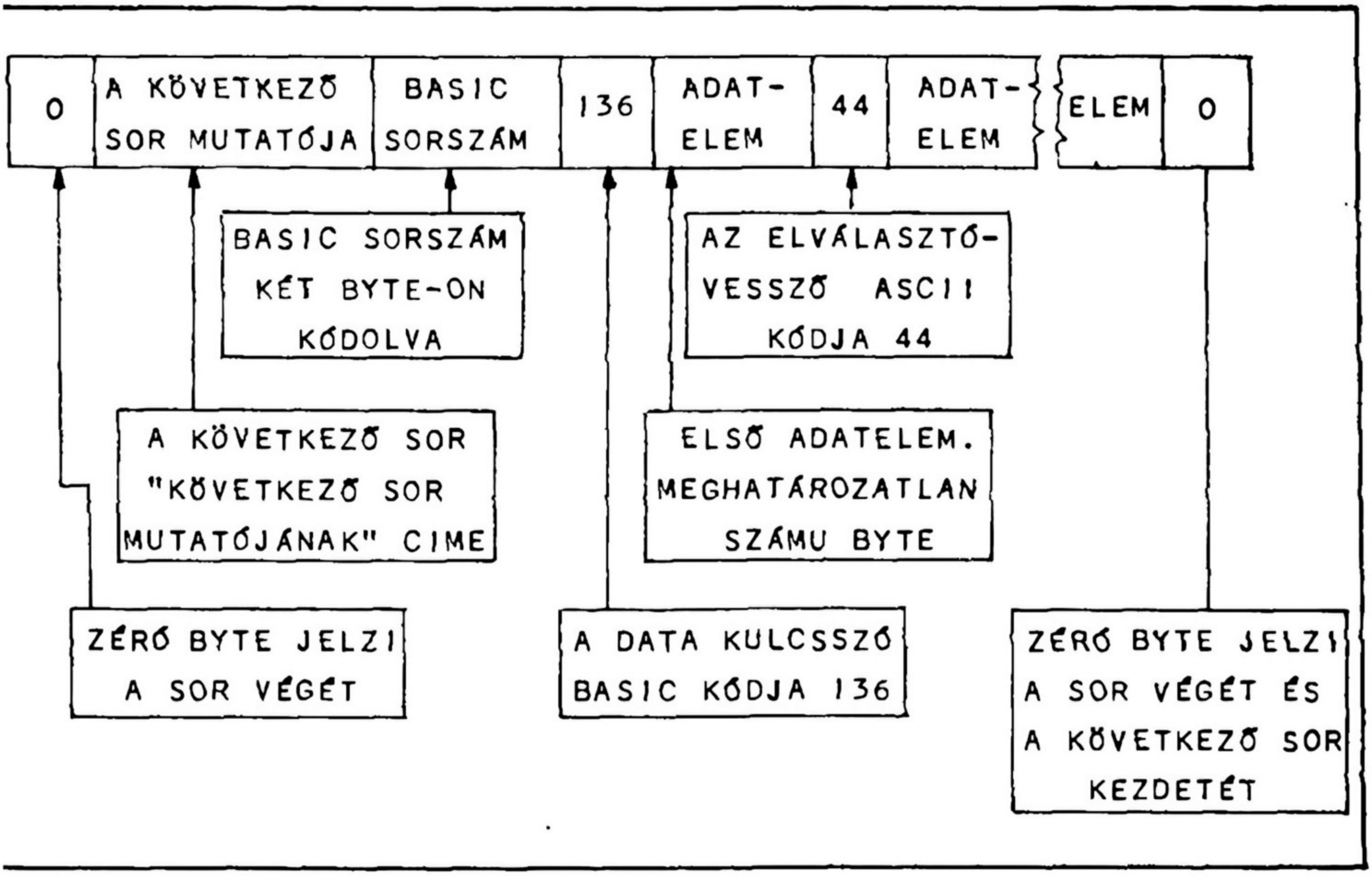
Általánosan használható kiutat jelent ebből a zavaros helyzetből, hogy létrehozunk egy nagyméretű változó vektort, és beleolvassuk a DATA összes elemét. Az egyes elemek elérése ezáltal könnyűvé válik. A rendelkezésre álló tár okoz gondot ennél a jellegzetes megközelítésnél. A programozó rendszerint oda lyukad ki, hogy kétszer annyi tárat használ, amennyit az adatok valójában igényelnek! Ez a fajta pocsékolás nem praktikus. Úgy tűnhet, hogy a BASIC korlátai ránk kényszerítik a nehézkes adatelérés elfogadását vagy kielégíthetetlen tárigényekkel találjuk magunkat szemben.

De szerencsére, a szükség találékonyságot szül! A BASIC végtére is csak egy program, és a memóriarekeszek vezérlik a működését.

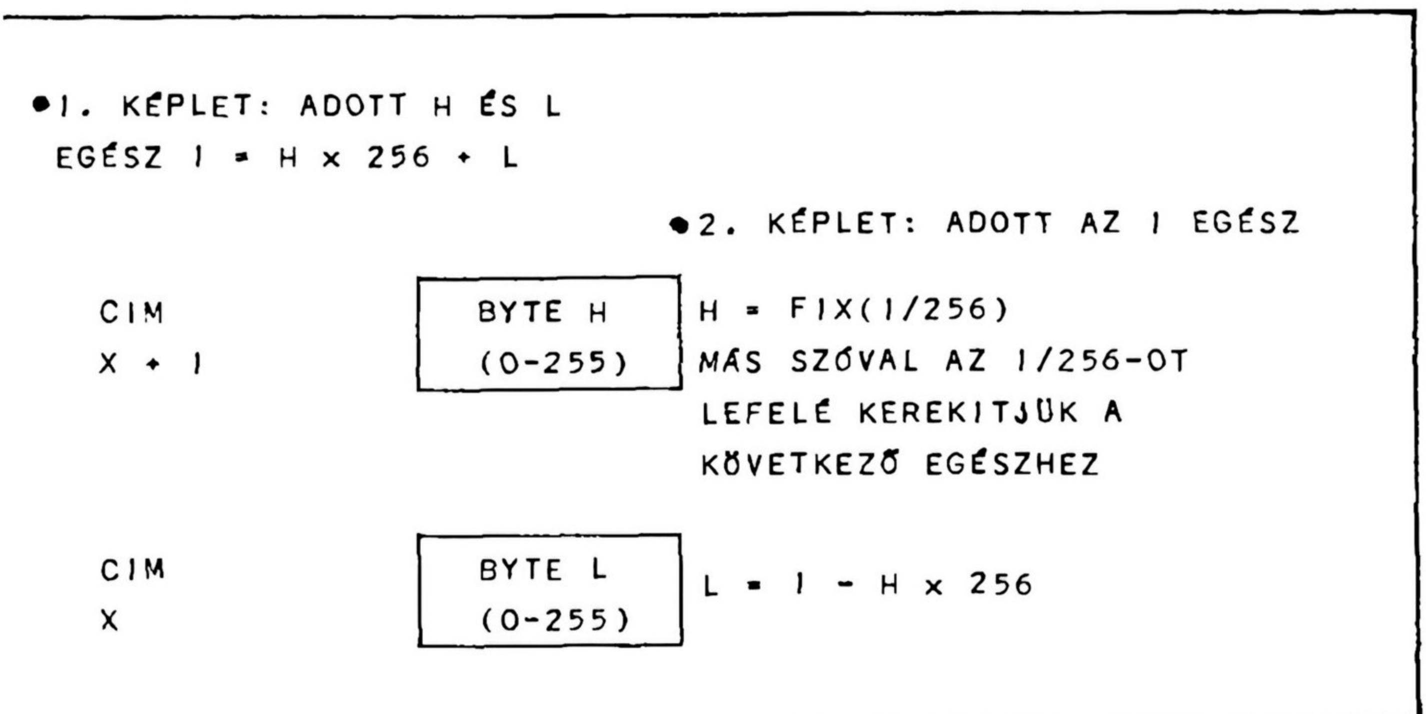
Módot kell találnunk arra, hogy kihagyhassunk DATA blokkokat, és azt olvashassuk, amit olvasni akarunk! Valahol a RAM-ban a BASIC fenntart egy futó mutatót, amely megmondja a READ parancsnak, hogy hol olvasson. Egy kis körültekintéssel megváltoztathatjuk a mutató értékét, úgy, hogy céljainknak megfeleljen. Ahhoz azonban, hogy ezt megtehessük, meg kell érteni, hogyan tárolódnak a DATA utasítások a memóriában!

A 3-3. ábra azt mutatja, hogyan van tárolva egy DATA utasítás a tárban. Figyeljük meg a formátum hat elemét:

- Egy nulla, amely elválasztja a DATA utasítást az előző utasítástól.
- A következő sor mutatója, amely a tár két byte-ján kódolva megadja a következő BASIC programsor következő sormutatójának tárcímét.
- BASIC sorszám, a sor sorszáma a tár két byte-ján kódolva.
- 136, ez a DATA szó kódja a BASIC-ben.
- Az elemek listája, vesszővel elválasztva, amely 44-ként jelenik meg a tárban.
- Egy nulla, ezt az utasítást választja el a következő BASIC sortól, és így tovább.



-3. ábra. Így tárolja a program a DATA utasítást és egyes részeit



-4. ábra. Egész számok tárolása két byte-ban

Kezdetben meg kell szokni a számok némelyikét, de néhány egyszerű konvenció érvényes. Először is, emlékezzünk rá, hogy minden BASIC program a 17384-től durván a 32767-ig terjedő RAM mezőben van tárolva. (Akinek Model I gépe van, ott ez a kezdőcím 17128.*) A 17384-es cím tartalma nulla, ez a fenti első elemnek felel meg. Azt jelzi, hogy BASIC programsor következik.

Aztán, a következő sor mutatója, ahogy ez a 3-4. ábrán látható, egy két tárhelyet elfoglaló kód. A címet úgy számítjuk ki, hogy a második hely tartalmát megszorozzuk 256-tal, és az eredményt az első hely tartalmához hozzáadjuk. Ennek a számnak a felhasználásával (természetesen ez 17384 és 32767 közé esik) a BASIC meg tudja állapítani, hol van a következő sor a tárban. Az A sor elején található következő sor mutatója megadja a soron következő B sor mutatójának helyét, és így tovább. A program legutolsó sorában a következő sor mutatója nullára van állítva, ez mint egy indikátor jelzi a program végét.

Mikor a BASIC először lesz felszólítva, hogy olvasson végig egy DATA utasítássorozatot, egy mutatót állít a legelső DATA utasítást megelőző nulla címére. Valahányszor egy READ utasítás hajtódik végre, ez a mutató előremozog, az éppen beolvasott adat mögé, a következő adatot megelőző vesszőre áll. Mikor egy sor utolsó adata is be lett olvasva, a mutató a sor végét jelző nullán van. A következő READ hatására a mutató előremozog, megkeresi a következő DATA sort és egy vesszőt, amin megáll. Az összes DATA sort így olvassa, amíg nem marad több adat. Ezután minden olvasási kísérlet hibaüzenetet vált ki. A tárban tehát az adatmutató mindig vagy vesszőre, vagy sor előtt álló nullára mutat.

A TRS–80 adatmutatója a 16639 és 16640-es tárcímeken található, a 3-4. ábra szerint kódolva. Ha a második számot 256-tal megszorozzuk, és hozzáadjuk az első számot, olyan címet kapunk, amely vagy egy vesszőé, vagy egy sor előtt álló nulláé.

A RESTORE parancs visszaállítja a mutatót 17384-re, a BASIC program legelején álló nullára.

Ha ismerjük a tárbeli címét annak a nullának, amely egy bizonyos DATA sort megelőz, az adatmutatóba kódolt formában beírhatjuk ezt a tárcímet. Ebben az esetben egy READ utasítás nem a program elejéről indul, hanem annak a DATA sornak első elemétől — függetlenül attól, hol is ez a sor. Képzeljük el! Pusztán azzal, hogy megváltoztatjuk az adatmutatót, akárhol elkezdhetünk olvasni, ha tetszik több száz elemet kihagyhatunk!

Csak az a probléma, hogyan találjuk ki ezeket a címeket? Nyolc DATA utasításblokk van. Az első három csak inicializálásra való, és csak egyszer olvassuk el, az utolsó öt azonban fontosabb. Hogyan találhatja meg a program ezek elejét?

* A HT-1080Z gépekre is ez érvényes. (*A fordító*)

Itt ismét hasznát látjuk a strukturált programozásnak. Ismerjük az öt fontos blokk sorszámát. Mint már láttuk, minden sor tartalmazza saját sorszámát, kódolt formában. Olyan programrészre van szükség, amelyik az inicializáló részben elhelyezve a következőket teszi:

- Egyesével megkeresi a DATA blokkok első sorát, azaz 5000-et, 6000-et, 7000-et, 8000-et és 9000-et.
- A későbbi hivatkozás számára egy numerikus vektorban tárolja ezt az öt igen fontos címet.

Ha az inicializálásnak ez a szakasza befejeződött, és a program valamelyik része hozzá akar férni valamelyik DATA blokkhoz, a megfelelő címet a vektorban megtalálja, levon belőle egyet, hogy a DATA sor előtti nullára mutasson, átalakítja a megfelelő két byte-os kóddá és az adatmutatóba teszi. A *Kardhalak és kincsek*-ben a vektort DA(*n*)-nek nevezzük az adatelérés (data access) után. Mivel minket az utolsó öt DATA blokk édekel, DA(*n*)-nek öt eleme van, DA(1)-től DA(5)-ig, amelyek a megfelelő mutatócímeket tartalmazzák.

A 3-5. ábrán látható a DA(*n*) vektort létrehozó inicializáló kód. Vegyük sorra parancsról parancsra, és lássuk, hogy határozza meg a megfelelő címeket!

Először néhány változó értékét előre beállítjuk. A P változót magára a tárcímre tartjuk fenn, és folyamatosan növeljük a megfelelő címértékig. P-t 17385-re állítjuk be a legelső BASIC programsor következő sormutatójának címére. (Emlékezzünk vissza, 17384-es címen van az első sort megelőző nulla!) Az N változót 1-től 5-ig növeljük, végiglépked a DA(*n*) vektor elemein.

Ezután ciklust szervezünk, amelyben az I változó 5000-től 9000-ig ezresével végighalad. Természetesen I megfelel a keresett sorszámoknak. Ne felejtsük, hogy minden BASIC sor sorszáma egy két byte-os kódban van tárolva a sor elején! A ciklusban az összes ilyen módon kódolt sorszámot át kell alakítani szokásos tízes számrendszerbeli értékre; ha ez az érték megegyezik I-vel, megtaláltuk a sort.

A 8-as sor első része ezt teszi. Mivel P mindig a következő sor mutatójának első byte-jára mutat, a sorszámot tartalmazó byte-ok P+2-n és P+3-on vannak. A konverziós képlettel az ezeken a címeken található értékekből elő-

```
6 P=16548:N=1:FORI=5000TO9000STEP1000
8 IFI=PEEK(P+2)+PEEK(P+3)*256THENDA(N)=P:N=
N+1:NEXTI:GOTO10:ELSEP=PEEK(P)+PEEK(P+1)*25
6:IFP=0THENCLS:PRINT"ERROR":END:ELSE8
10 CT(0)=1:CT(12)=RND(10)+19:CLS
```

3-5. ábra. Ez az inicializáló kód tölti fel a DA(n) vektort a fontos DATA blokkok címeivel

állítjuk az eredeti sorszámot és összehasonlítjuk I-vel. Ha megtaláltuk a sort, P jelenlegi értékét a vektor DA(N) elemébe mentjük; N-et megnöveljük, hogy így a következő sor címét a vektor következő elemébe mentsük. A ciklus folytatódik, majd ha készen vagyunk, kilépünk belőle.(A 10-es sor az inicializáló eljárás folytatása.)

Nyilvánvaló, hogy ez a programrészlet nem az 5000-es sorral találkozik először. Mi történik, ha a sorszám nem egyezik meg I-vel? Ebben az esetben az ELSE-t követő BASIC kód hajtódik végre. P a most következő sor mutatójára mutat; most átállítjuk a mutató értékére a P-t. A P, P+1 című byte-ok tartalmát tízes számrendszerbeli számmá alakítjuk át, és az eredményt a P változóba tesszük. Most már folytatódhat a keresés, mivel P a következő BASIC sorra mutat. A programrészlet addig ismétli a 8-as sort, amíg sorszámegyezés nem következik be.

Foglaljuk össze még egyszer! A P változó felhasználásával a programrészlet egyesével végighalad a BASIC sorokon, a sorok elején található következő sormutató segítségével. Minden sorban megnézi a kódolt sorszámot, megpróbálja az 5000-es sort megtalálni. Mikor megtalálja, P értékét DA(1)-be menti. Ezután az eljárást megismétli 6000-től 9000-ig. Végül a DA(*n*) vektor tartalmazni fogja azt az öt tárcímet, amelyekkel megtalálhatjuk az öt legfontosabb DATA blokkot.

Az adatok elérését biztosító szubrutin

Az inicializáló programrésznek ez a fele ellátja az adatkiolvasás munkájának felét: megadja minden DATA blokk kezdetét. Szükség van még egy olyan szubrutinra, amely ennek felhasználásával megkeresi az egyes elemeket. Az összes DATA blokk formátuma alapvetően azonos: DATA sorokból áll és a DATA sorok elemekből. A szubrutinnak a következő feladatai vannak:

- Találja meg a megfelelő DATA blokkot a DA(*n*) vektor segítségével.
- Találja meg a blokk megfelelő sorát.
- Állítsa a DATA mutatót erre a sorra.

Ha már a DATA mutató be lett állítva, a főprogram READ parancsokkal találja meg a soron belül a keresett elemet. (Ekkor már esetleg szükség lehet arra, hogy READ ciklussal lépjen át néhány elemet; az átlépés nagy részén már túl vagyunk időt rabló ciklusok nélkül.)

Ez a három követelmény szükségessé teszi két változó értékének beállítását, mielőtt a szubrutin elvégezhetné a feladatát: be kell állítani a blokk és a sor sorszámát. A blokk sorszáma 1 és 5 közötti szám, a sorszám 1-től a kiválasztott blokkban levő DATA sorok számáig terjedhet.

A 3-6. ábrán látható az ACCESS* szubrutin kódja. (Hasonlóan az összes

*ACCES itt elérés, hozzáférés. (*A fordító*)

kalandszubrutinhoz az 1000 és 1999 közti BASIC sorokon található.) A főprogram csak azután hívja meg a szubrutint, hogy két változót beállított: az A változóba kerül a DATA blokk száma, B-be a sor blokkon belüli száma. ACCESS lefutása után, a következő READ parancs az A blokk B sorában kezd.

Access először megkeresi az A DATA blokk elejének tárbeli címét, a DA(N)-ben tárolt számok segítségével. A P változóba kerül ez a cím. Emlékezzünk rá, hogy ez a cím a blokk első DATA utasításának következő sormutatójára mutat!

Mi történjen, ha a B-ben tárolt szám 1, azaz mi történjen, ha a keresett sor a blokk első sora? Ha ez a helyzet, P máris a megfelelő sorra mutat, és a szubrutin az 1042-es sorra ugrik, ahol beállítja a DATA mutatóját. Ha nem ez a helyzet, a megfelelő sort meg kell keresni. Hasonló módszert használunk, mint az inicializáló programrészben. A következő sor mutatóját kiolvassuk, és a P változóba tesszük. Minden ilyen alkalommal egy sort lépünk át és P a következő sorra mutat. Az átlépést egy 1-től (B–1)-ig futó ciklussal oldjuk meg; a ciklus átlépi az érdektelen sorokat, amíg P-be nem kerül a kívánt sor címe. Ily módon a keresett sort igen gyorsan megtaláljuk.

```
 NEV:            ACCESS
 TIPUS:          SZUBRUTIN
BEMENO ADATOK:   A=adatblokk száma
                 B=DATA sor sorszáma
EREDMENY:        A DATA mutato a sor
                 elejére áll

1040 P=DA(A):IFB=1THEN1042ELSEFORZ=1TOB 1:P
=PEEK(P)+PEEK(P+1)*256:NEXTZ
1042 P=P-1:POKE16640,FIX(P/256):POKE16639,P
-FIX(P/256)*256:RETURN
```

3-6. abra. Az Access szubrutin

Most már csak az van hátra, hogy úgy állítsuk be a DATA mutatót, hogy a READ utasítás helyesen olvassa a sort. Talán emlékszik rá az olvasó, hogy normális működési módban a DATA mutatónak a DATA sor előtti nullára kell mutatni, hogy a READ utasítás a sor első adatával kezdjen. Nos, P már a DATA sor következő sormutatójára mutat, és a nulla jelzőbyte pont egy byte-tal előbb van. Ha a DATA mutatót P mínusz 1-re állítjuk, pont úgy lesz beállítva, ahogy a Microsoft BASIC rendesen beállítja — és a READ működik. Annak a képletnek a felhasználásával, amivel számokat kódoltunk két byte-ba,

P-t konvertáljuk, a 16639 és 16640-es tárbeli címekre tároljuk, ezek alkotják a DATA mutatót. Ez minden!

Az Access szubrutin nagyon felgyorsítja a dolgokat. Pl. ha szükségünk van a 7-es helyiség rövidített leírására, az eljárás egyszerű. A helyiségek leírása az 5-ös DATA blokkban van, ezért A-t 5-re állítjuk be. A helyiség sorszáma megfelel a sor sorszámának, ezért B-t 7-re állítjuk. Ezután hívjuk meg Access-t (GOSUB 1040). Ha befejeződik, hajtsunk végre két READ utasítást; mivel mindkettő ugyanabban a DATA sorban van, az egyik a bővebb leírást olvassa be, a másik a rövidítettet. Így ravaszul fértünk hozzá az adathoz, gyorsan és hatékonyan, anélkül, hogy el kellett volna olvasni az összes megelőző adatot.

Mivel minden kalandprogram alapjában véve adat-visszakereső és -módosító program, nem hathat meglepetésként, hogy több szubrutin még közvetlenebb módon kapcsolódik az egyes DATA blokkokban tárolt adatokhoz. Ezek a szubrutinok arra használják az Access-t, hogy megtalálják, amire szükségük van. Az Access-t nevezhetnénk tehát (bár ezt nehéz túlélni!) szub-szubrutinnak. Ez a titka minden igazán bonyolult programnak: egyszerű programrészek látnak el egyszerű feladatokat, más programrészek az egyszerűbbek segítségével bonyolultabb feladatokat látnak el, és így tovább felfelé.

Látni fogjuk, hogy ennek eredményeként a *Kardhalak és kincsek* főprogramja, a Vezérlő, ténylegesen rövid és tetszetős. Hogy lehet ez? Úgy, hogy a munka részleteit átruházza az alacsonyabb szinten található szubrutinrétegre, amely egyfajta kollektív végrehajtó program.

Most, hogy rendelkezésünkre áll egy hatékony módszer adatok elérésére, az érdekel bennünket, hogy mely szubrutinok használják ezt a módszert. Lássunk néhányat!

Vegyük elő az üzenetet!

Az üzenetek kiírása az egyik funkció, amely gyors adatelérést használ. Az egész 3-as blokk üzenetek tárolására van szánva. Megkönnyíti a dolgokat, ha az üzenetekhez számokat rendelünk; mikor szükség van egy üzenet kiírására, az Access-t használjuk fel a megfelelő üzenet kikeresésére.

A Mesprt* szubrutint hívjuk meg, ha üzenetek kiírására van szükség. A 3-7. ábra megadja ennek a szolgáltatásnak a BASIC kódját, amely a program szubrutinrészében, az 1100-as sorban van. Mesprt csak egy adatot kér: az üzenet sorszámát 1 és az üzenetek maximális száma között. Innen kezdve Mesprt mindent magára vállal, megkeresi az üzenetet (természetesen Access segítségével) és kiírja a képernyőre.

*Üzenet nyomtatása. (*A fordító*)

Az a program, amely üzenetet akar kiíratni, B-be beállítja az üzenet sorszámát és meghívja Mesprt-t (GOSUB 1100). Ha az Access-t fel akarjuk használni, emlékezzünk rá, hogy Mesprt-nek két információt kel szolgáltatnia: a blokksorszámot és a sor számát. A sor száma egyszerű, mivel az üzenet száma azonos a sor számával — az üzenetek tárolójában minden sor egy üzenetet tartalmaz. Ez a szám már B-ben van, Access szerint is pont ott a helye. A blokk száma sem okoz gondot. A speciális üzenetek a 3-as blokkban vannak. Ezért Mesprt A-ba hármat tesz, pont úgy, ahogy Access elvárja. Mesprt meghívja Access-t. Mesprt-nek ezután egy READ utasítást kell csak végrehajtania, és meg is van az üzenet. Az üzenet be lett olvasva, ki lett írva, itt a vége, fuss el véle!

A Mesprt használata valóban megszabadítja a kalandprogram íróját az üzenetek számontartásától. Lehet látni olyan programokat, amelyekben szerteszét vannak üzenetek, némelyik többször is.

```
NÉV:               MESPRT
TIPUS:             SZUBRUTIN
BEMENŐADAT:        B=Az uzenet sor száma
EREDMÉNY:          Kikeresi és kiírja
                       az üzenetet

1100 A=3:GOSUB1040:READA$:PRINTA$:RETURN
```

3-7. ábra. A Mesprt szubrutin

A Mesprt használtával a programozó egy kupacba gyűjti az üzeneteket, és számmal hivatkozik rájuk. Ezzel munkát takarít meg, és mint tudjuk, a programozó örül minden segítségnek, amihez potyán jut hozzá!

A játékos mozgásának nyomonkövetése

Egy kalandprogram leggyakrabban használt parancsai a mozgási parancsok. A játékos — ahogy helyiségről helyiségre halad a helyszínen — elsősorban felfedező. A programozó arra törekszik, hogy ezek a parancsok rövid idő alatt hajtódjanak végre, de sok minden történik, mikor a játékos mozogni próbál. Időbe telik, míg megállapítjuk, melyik helyiségbe jut el, ha egy megadott irányban mozdul. Nyilvánvaló, hogy a bejárási táblázat néven ismert adatblokk

igénybevétele nagyon nagy. Access segítségével előáshatjuk a szükséges helyiség számát, de célszerűbb, ha van egy kissé magasabb szintű szubrutin, amelyet kifejezetten arra terveztek, hogy a leghatékonyabban férjen hozzá a bejárási táblázathoz.

Hozzuk létre a 3-8. ábrán látható szubrutint, nevezzük Travec-nek, mivel azokat a bejárási irányokat határozza meg, amelyek egyes elmozdulások végső állomásai. Travec az 1120-as sorban van, a program szubrutin területén. Elsődleges célja, hogy céladatokat szolgáltasson a bejárási táblázatból, ha adott a jelenlegi helyiség száma és a kívánt mozgásirány.

Emlékszik még az olvasó a bejárási táblázat szerkezetére? Minden egyes helyiséghez egy sor tartozik az eredő végállomásokkal. Minden sorban tizenegy elem van, az első tíz megfelel a tíz lehetséges elmozdulási iránynak, a tizenegyedik a feltehető legvalószínűbb irány, arra az esetre, ha a játékos nem egyértelmű kifejezést használ (pl. LÉPJ BE). A végállomás eléréséhez a Travec először is megkeresi azt a DATA sort, amely a játékos pillanatnyi tartózkodási helyéhez tartozik. Azután átolvassa a sort a szükséges iránynak megfelelő elemig.

```
NÉV:                  TRAVEC
TIPUS:                SZUBRUTIN
BEMENŐ ADAT:          D=a szándékolt moz-
                      gás iránya (1-11)
EREDMÉNY:             A=a szándékolt moz-
                      gás végcélja

1120 B=CT(0):A=1:GOSUB1040:FORY=1TOD:READA:
NEXTY:RETURN
```

3-8. abra. A Travec szubrutin

A Travec Access segítségével jut adathoz. Az Access-nek két információra van szüksége: a B változóban legyen a sor blokkon belüli száma, és az A változóban legyen a DATA blokk száma. A bejárási táblázat használata során a jelenlegi helyiség száma a fontos adat. A sor száma megegyezik a helyiség számával, ezért a Travec B-be teszi be a jelenlegi helyiség számát. (A CT(0) változó tartalmazza a játékos jelenlegi helyét.) A bejárási táblázat az 1. az Access által kezelt öt blokk közül; ezért Travec A-t 1-re állítja. Az Access-t úgy használjuk fel, hogy végrehajtjuk a megfelelő GOSUB utasítást.

Amikor az Access befejeződik, a Travec tudja, hogy a megfelelő információt tartalmazó sort olvashatja, de tudnia kell, hogy a 11 elem közül melyiket keresse ki. Ebből a célból a Travec-et hívó program még egy változót szolgáltat, D-t, ez mondja meg a mozgás irányát.

Az 1 és 8 közötti mozgásirányok az égtájaknak felelnek meg, 9 jelenti a felfelé, 10 a lefelé irányt, 11 a legvalószínűbb feltételezhető irányt. Ha ez a szám D-ben van, Travec pontosan tudja, meddig kell olvasnia. Lefuttat egy rövid READ ciklust 1-től D-ig számlálva. Mikor a ciklus lefutott, a végállomást jelölő szám az A változóban van.

A *Kardhalak és kincsek*-ben még sok más szubrutin használja az Access-t, hogy az adathoz hozzáférjen. Ezeket későbbi fejezetekben ismertetjük részletesen, amint szükségünk van rá.

Adatok egészekbe préselése

A szisztematikus programozó sohasem pazarló. Mindig keresi az információtárolás ügyesebb módját, mindig arra törekszik, hogy összesűrítse, tömörítse és összekombinálja az adatokat. A rendelkezésre álló tár korlátai miatt (a 16K legnagyobb részét szöveg foglalja el) mi sem engedhetjük meg, hogy elszalaszszunk egy jó adattömörítési módot. Ha a Microsoft BASIC-re hagyjuk a döntést, akkor már a numerikus változók létrehozása elnyeli a memória maradékát, és egy szép, kövér OM ERROR-t ad cserébe.

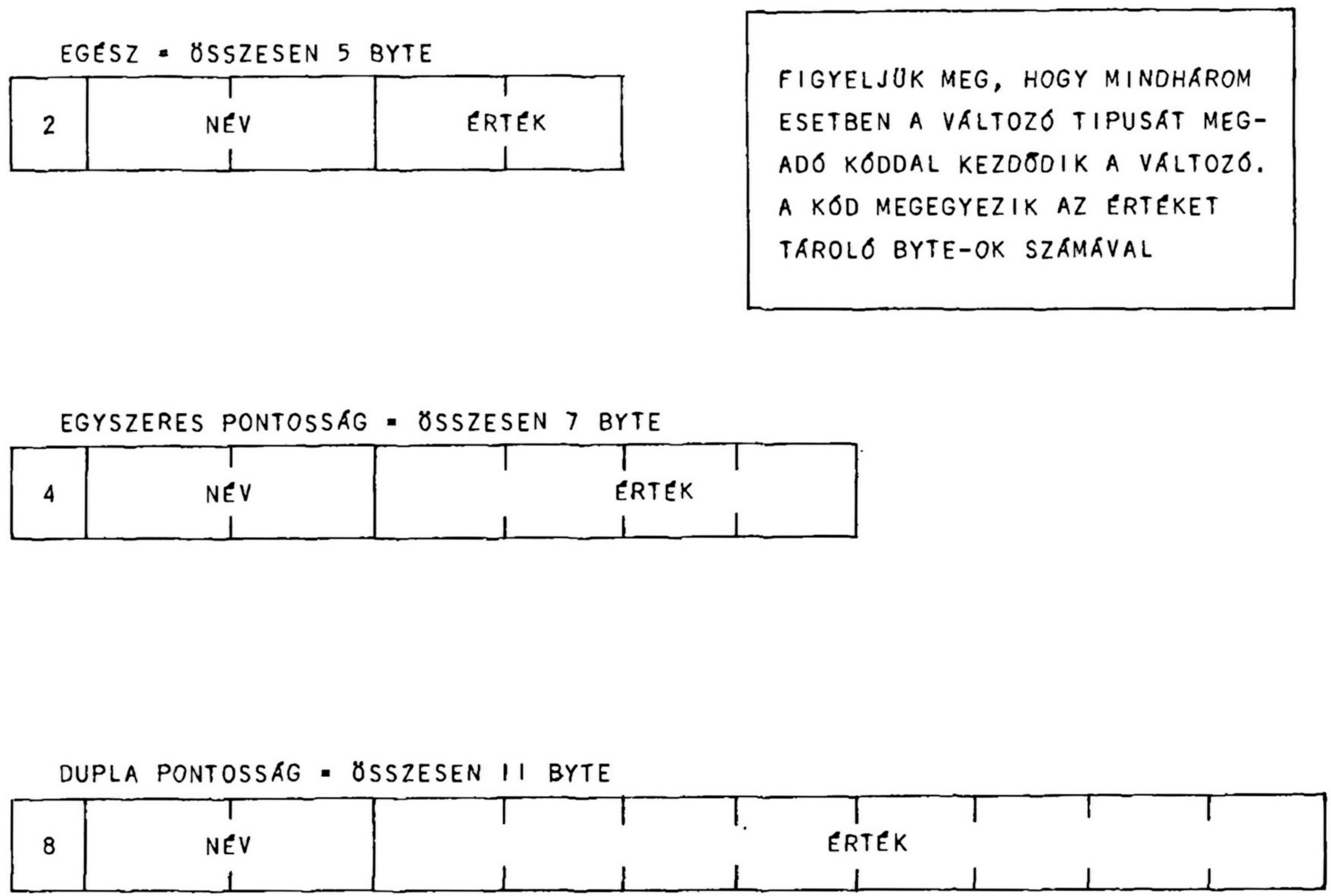
Hátra van még a program strukturálása során alkalmazott módszertan utolsó elemének megbeszélése; ez a változók szervezése. Néhány gondolat előrebocsátása ebben a tekintetben sok nehézségtől kímél meg, ha majd végre leírjuk a RUN-t és lenyomjuk az ENTER-t.*

Először essünk túl néhány egyszerű előkészületen! A kalandprogram írójának elsősorban valamilyen rendszert kell bevezetnie a változó nevek megválasztásában, miközben a programrészleteket írja. Ha nem emlékezünk, melyik változót használtuk utoljára, ne használjunk egyszerűen egy újat! Mert az lesz az eredménye, hogy a BASIC programot teleszórjuk változókkal A-tól Z-ig, holott sok esetben egy tucatnyi is elég lenne.

Miért jelent ez gondot? Azért, mert valahányszor bevezetünk egy új változót, a Microsoft BASIC helyet foglal neki a tárban. Minden használt változónak lefoglal három byte-ot, épp csak a saját nyilvántartása részére, ebben még nincsenek benne a változó aktuális értékét tartalmazó byte-ok. Ezeket a lekötött byte-okat nem használja fel; egyszerűen csak ott csücsülnek, pocsékban.

Helyes gyakorlat az, hogy papíron tartjuk nyilván, milyen változót és milyen célra használunk. Valahányszor szükségünk van egy változóra, kény-

*A HT-1080Z-n NEW LINE. (*A fordító*)



3-9. ábra. A három számtípusú változó és az általuk elfoglalt tár mérete

szerítsük magunkat, hogy megvizsgáljuk a listát: nem használhatunk-e fel egy korábban létrehozott változót! A legjobb példa az ilyen típusú szervezettségre a FOR-NEXT ciklus. Elképzelhető, hogy néhány kijelölt változóra korlátozhatjuk ezeket, pl. I, J és K, amelyek egyedüli célja, hogy újra meg újra ciklusokban használjuk fel őket. Álljunk ellent a kísértésnek, hogy betűről betűre ugráljunk.

A második betartandó ökölszabály a felhasznált változók típusával kapcsolatos. Végül is nem minden változó születek egyformának. Nézzük a 3-9. ábrát, és hasonlítsuk össze az érintett byte-ok számát! A változótípusok között a leghatékonyabb az egész; a számokat olyan kétbyte-os kódba préseli bele, amilyent már használtunk. Az egészek pontossága csekély, mivel törtrész nincs megengedve; erre a célra hozták létre az egyszeres és kétszeres pontosságú változókat. Rendszerint azonban a pontos változók a matematikai jellegű programoknak valók. A kalandprogramoknak nincs szüksége hajmeresztő pontosságra.

Vigyázat, ha nem mondjuk meg a BASIC-nek, hogy mire van szükségünk, többet ad, mint amire számítunk! Eleve egyszeres pontosságú változóval dolgozik. Vagyis: ha nem mondjuk meg, hogy milyen típusú változóra van szükségünk, a BASIC feltételezi, hogy egyszeres pontosságot akarunk. Ez változónként néhány többletbyte-ot eredményez — 40 százalék többlet a változó tárolóhelyében! Kell, hogy legyen jobb megoldás!

Természetesen, ha akarjuk, minden alkalommal specifikálhatjuk az „egészet", csupán egy százalékjelet (%) kell a változó neve után illeszteni. Csakhogy ez elég kényelmetlen, és ráadásul a BASIC hatékonyabb módszert is biztosít, amivel nem marad ki véletlenül sem egy változó specifikálása. Nos, ez a DEFINT utasítás.

Ha a DEFINT utasítást használjuk a program inicializáló részében, ezzel elérjük, hogy bizonyos változók eleve egész típusúak legyenek.

A *Kardhalak és kincsek* az utasítás legátfogóbb formáját használja: DEFINT A–Z. Ez lényegében azt mondja a BASIC-nek, hogy minden numerikus változót, amely A és Z közötti betűvel kezdődik (ez az összes numerikus változót jelenti) kezeljen egészként. Végeredményben azt az üzenetet közöltük a BASIC-kel: „Érzékenyek vagyunk a helyfoglalásra, takaríts meg helyet."

Használjunk ki minden számjegyet!

Az előző megjegyzések a változók megválasztásáról a józan észen alapulnak, nincs bennük semmi új. Most fogjuk meg „trükkösebben" a dolgot! Már korábban láthattuk, hogy egy egész tekintélyes mennyiségű információt tárolhat. A Microsoft BASIC-ben egy egész a −32768-tól +32767-ig terjedhet. Mi olyan számokat tárolunk, amelyek ritkán nagyobbak 10-nél, és szinte mindig kisebbek

100-nál. Hasznunkra válik ilyen körülmények között, ha mindent kifacsarunk egy egészből, amit csak lehet.

Ennek az egyszerű egésznek hat olyan tárolóterülete van, amely BASIC-ből könnyen elérhető. Van öt számjegy (ezeket jobbról balra 1. számjegytől 5. számjegyig számozzuk) és egy hely az előjelnek (amely vagy pozitív, vagy negatív). Természetesen vannak bizonyos korlátozások arra vonatkozóan, hogyan használhatjuk ezt a hat területet. Egyik számjegyhez sem rendelhető olyan érték, hogy az összeállított egész végső értéke meghaladja az itt megadott határokat. Így az 5. számjegy sohasem lehet 3-nál nagyobb; mindig 0 és 3 közötti kell legyen. A másik négy számjegy tetszőleges számjegy 0 és 9 között, de az 5. számjegy jó, ha 3-nál kisebb, mert a teljes egész nem haladhatja meg a 32767-et. Ha a programozó ezt a fajta adattömörítést használja, úgy kerülheti el leginkább ezt a veszélyt, hogy olyan funkciót rendel az 5. számjegyhez, amelyik soha nem lépi túl a 2-t. Ellenkező esetben gondosan ügyelnie kell a többi számjegyre.

Az előjel csak kis mennyiségű információt hordoz, mivel mindössze két állapot egyikében lehet. Ez mégis hasznos, és az előjel valójában nincs hatással az őt követő számra. Ezen túl, mint látni fogjuk, az előjelet sokkal könnyebb megvizsgálni és megváltoztatni, mint az egész egyes számjegyeit.

Milyen módszert használhatunk, ha feltesszük, hogy egy egész helyiértékein kis számokat akarunk tárolni? Nincs olyan BASIC utasítás, amelynek az lenne a rendeltetése, hogy közvetlenül megváltoztassa egy szám számjegyeit. Ezért magunknak kell valamilyen programrészt készíteni — egészen pontosan kettőt. Az egyik szubrutin vesz egy egészet, felbontja öt számjegyre, és a számjegyeket különálló változókban tárolja, olyan helyen, ahol könnyű őket megvizsgálni és megváltoztatni. A másik szubrutin veszi az öt különálló változó értékét és megfordítja a folyamatot: ismét egy teljes egészet rak össze belőlük.

Egy egész szétszedése

Az egészeket számjegyeire bontó szubrutint nevezzük Analyz-nek, kódja a 3-10. ábrán látható. A CT(5) és CT(11) közötti változók a vizsgált egészhez vannak kapcsolva. Mikor a szubrutin befejeződik, a számjegyek az elsőtől az ötödikig a CT(6) és CT(10) közötti változókban tárolódnak. Ezenkívül az egész előjele bekerül CT(11)-be; értéke 1 pozitív és −1 negatív egész esetén.

Egy FOR-NEXT ciklust futtatunk le, hogy nullára állítsa a CT(6) és CT(10) közötti változókat. Erre azért van szükség, mert egy korábbi egész felbontás számjegyei ott maradhatnak és megzavarhatják az eredményt. Ezután az egészt olyan alakra kell hozni, amelyben az egyes számjegyek elkülöníthetők. Mivel CT(5) numerikus változó, nem tanulmányozható számjegyenként; nincs olyan BASIC utasítás, amely ezt megtenné. Ha CT(5) tartalmát karakterláncformára alakítjuk, a Microsoft BASIC hatékony szövegkezelő utasításaival részeire bonthatjuk.

```
NÉV:           ANALYZ
TÍPUS:         SZUBRUTIN
BEMENŐ ADAT:   CT(5)=Egy egész
EREDMÉNY:      CT(6)...CT(10)=az egész
               számjegyei és CT(11) az
               előjel

1000 FORZ=6TO10:CT(Z)=0:NEXTZ:B$=MID$(STR$(
CT(5)),2):FORZ=1TOLEN(B$):CT(6+LEN(B$)-Z)=V
AL(MID$(B$,Z,1)):NEXTZ:IFCT(5)<0THENCT(11)=
-1:RETURN:ELSECT(11)=1:RETURN
```

3-10. ábra. Az Analyz szubrutin

Ezt az átalakítást az STR$ függvény végzi. Az átalakítás során a teljes szám karakterláncformára lesz átalakítva — az előjelet is beleértve! Pillanatnyilag csak az öt számjegyet akarjuk elkülöníteni, a szám előtt álló előjel csak zavart okoz. A MID$ függvény segítségével leválasztjuk az új lánc első karakterét. (Az első karakter szóköz, ha a szám pozitív, mínuszjel, ha a szám negatív.)

Az STR$ CT(5)-öt karakterlánccá alakítja. Ebből a MID$ újabb láncot hoz létre, amely a második karakternél kezdődik. Ezután ezt a láncot B$-ként tároljuk a tárban. Most már egy FOR-NEXT ciklussal karakterenként elemezhetjük B$-t. Az utolsó karakter az 1. számjegy, és a számjegysorszám jobbról balra nő. Ne felejtsük el: nem biztos, hogy a szám ötjegyű, ez a szám értékétől függ!

A FOR-NEXT ciklus 1-től a láncot alkotó karakterek számáig fut; LEN határozza meg a végértéket. A karakterláncot balról jobbra értékeljük ki, ismét a MID$ függvényt használjuk. Ahogy Z növekszik, MID$ sorra kiválasztja az összes karaktert. VAL a korábbi STR$ függvény ellenkezőjét csinálja; a kiválasztott karaktert számértékké alakítja át, hogy CT(*n*) változóban tárolhassuk.

Az egyenlet bal oldala biztosítja, hogy a megfelelő érték a megfelelő változóban kössön ki. Legyen pl. Z egyenlő 1-gyel. A számjegy a karakterlánc bal szélén áll. De melyik is, az 5. számjegy, a 4.? Mindez a karakterlánc hosszától függ; ezért a LEN függvénynek fontos szerepe van. Ötjegyű szám esetén bal szélső számjegy CT(6+5−1), azaz CT(10)-be kerül, az 5. számjegynek megfelelő változóba. Ennek így is kell lennie.

Miután a ciklus különálló változókba pakolta a számjegyeket, az utolsó

feladat az előjel tárolása. Ha CT(5) nullánál kisebb, −1 kerül a CT(11)-be, egyébként 1.

Az eredmény? Ha egy program számára jelentős egy tárolt változó, mondjuk a 3. számjegy, akkor egy GOSUB 1000 segítségével meghívja Analyz-t, azután megvizsgálja CT(8)-at. Így az élet sokkal könnyebb!

Egy egész ismételt összerakása

Most tételezzük fel, hogy egy program felhasználta Analyz-t, hogy ellenőrizzen egy számjegyet, és most meg akarja változtatni ezt. Szükségünk van tehát egy szubrutinra, amely fogja az összes számjegyet a megváltoztatottal együtt, és ismét összerakja őket, új egészet hozva létre. Az Analyz-nek ezt az ellentettjét kereszteljük Synthe-nek, listája a 3-11. ábrán látható. CT(5)-be tölti azt az egész számot, amely a CT(6) és a CT(10) között található öt számjegyből jön létre, abban az esetben is, ha némelyik számjegy zérus. A változó előjelét attól függően állítja be, hogy CT(11)-ben +1 vagy −1 van-e.

```
NÉV:               SYNTHE
TIPUS:             SZUBRUTIN
BEMENO ADATOK:     CT(6)...CT(10) egy
                   egész számjegyei,
                   CT(11) az elöjel
EREDMÉNY:          CT(5)= az egész

1020 CT(5)=CT(10)*10000+CT(9)*1000+CT(8)*10
0+CT(7)*10+CT(6):CT(5)=CT(5)*CT(11):RETURN
```

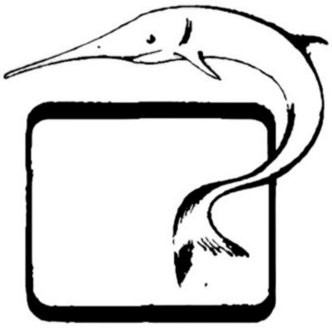
3-11. ábra. A Synthe szubrutin

Az egész eljárás Analyz-hez nagyon hasonló is lehetne: szövegkezelő függvényekkel a számjegyeket karakterekké alakítjuk, aztán összeláncoljuk őket, majd az új láncot ismét konvertáljuk numerikus formára. A 3-11. ábrán bemutatott módszer azonban gyorsabb és egyszerűbb.

Végül is, valójában minden számjegy a szám egy helyiértékét jelenti. Az 1. számjegy az egyes helyiérték, a 2. számjegy a tízes és így tovább. Ezért a Synthe minden számjegyet megszoroz a helyiértéknek megfelelő együtthatóval és az eredményeket összeadja. Ezután CT(11)-et szorzóként használva beállítja az előjelet. A végeredményt CT(5)-ben tárolja, és ezzel teljesen körbejártuk az egészek kezelését.

Most teszünk egy lépést lefelé

Befejeztük azoknak a finom szerkezeti részleteknek a megtárgyalását, amelyek jelentős szerepet játszanak egy feszes, hatékony kalandprogramban. Legfőbb ideje, hogy felnyissuk a nyikorgó csapóajtót és belépjünk a homályba. Hogyan jelenítjük meg a helyiségek leírását? Mi van a tárgyakkal? Mi van az ellenségek támadásával? Ez mind része a kalandprogram fő végrehajtó szekciójának, és mindezt a következő fejezetben fejtjük ki.



4. FEJEZET

Leereszkedünk a föld alá

Egy tipikus kalandprogram valamilyen képes útibeszámolóhoz hasonlítható, ami elénk vetíti a környezet képét, amint a kalandozó jár-kel. Igazában a programnak két működési állapota van. Az első állapotában helyiségeket, tárgyakat és hasonlókat ír le; a program alszik és parancsra vár. A másik állapotban parancsot írnak be, ez behív egy kezelőt, és valamilyen eredmény jön létre. Rendesen a kalandprogram szabályos ciklusban fut e két állapot között.

Mielőtt az első állapot beállna, a programnak némi előkészületre van szüksége. Az inicializálást részben már tárgyaltuk az előző fejezetben. Mielőtt leereszkednénk a föld alá. tegyük teljessé az előkészületekről alkotott képünket!

Írjuk le a RUN, és nyomjuk le az ENTER* billentyűt!

A 4-1. ábrán látható a *Kardhalak és kincsek* teljes inicializálási folyamata. Mikor leírjuk, hogy „RUN", és lenyomjuk az ENTER billentyűt, ezek a sorok hozzák létre a fő vezérlőrész futásának alapvető keretét.

Kezdjük a legelején! Egy játékprogram nem igazán szellemes címfelirat nélkül. Sajnos nem engedhetjük meg. hogy a drága tárat olyan hasznos dolgokra fordítsuk, mint a játékszabályok vagy a játékkal kapcsolatos tanácsok. Be kell érni a címmel. A CHR$(23) 32 karakteres módba váltja át a képernyőt, nagy, figyelemfelkeltő betűket eredményez. A PRINT@ utasítások biztosítják, hogy a cím sorai éppen oda kerüljenek, ahova akarjuk.

Figyelmeztetnünk kell azokat a felhasználókat, akik nem járatosak a 32 karakteres kiírási módban. Ilyenkor a betűk szélessége megduplázódik, és a képernyő-memóriában minden második byte kimarad. Ezért a PRINT@

* A HT-1080Z-n NEW LINE billentyű van.

```
2 CLS:PRINTCHR$(23):PRINT@466,"Köszöntjük a
":PRINT@522,"KARDHALAK ÉS KINCSEK":PRINT@59
7,"játékban!"
4 CLEAR500:DEFINTA-Z:DIMTX$(4),DA(5),RM(20)
,OB(16,1),BK(10),CT(12):FORI=1TO20:READRM(I
):NEXT:FORI=1TO16:READOB(I,1),OB(I,0):NEXT:
FORI=1TO10:READBK(I):NEXT
6 P=16548:N=1:FORI=5000TO9000STEP1000
8 IFI=PEEK(P+2)+PEEK(P+3)*256THENDA(N)=P:N=
N+1:NEXTI:GOTO10:ELSEP=PEEK(P)+PEEK(P+1)*25
6:IFP=0THENCLS:PRINT"ERROR":END:ELSE8
10 CT(0)=1:CT(12)=RND(10)+10:CLS
```

4-1. ábra. Az inicializáló rész

utasítást csak úgy szabad használni, hogy páros sorszámú képernyőhelyeket címezzen meg! Kísérletképpen próbáljuk ki a PRINT@ -ot páratlan számmal; a memória tárolja a szót, de a képernyő nem hajlandó megjeleníteni.*

Ezután néhány adminisztrációs feladat következik a számítógépen belül. Jó néhány a tárkiosztással kapcsolatos. A 4-1. ábrán látható, hogyan kezeli a 4-es sor ezeket az igényeket.

Pl. tudatni kell a géppel, hogy mekkora tárterületet tartson fenn a karakterláncok létrehozására és tárolására.

Talán tudja az Olvasó, hogy bekapcsoláskor a BASIC azonnal lefoglal 50 byte-ot karakterláncok, szövegek tárolására; ez a terület a tár magas címein található a tárméret határának közelében. Ennél azonban többre van szükség. Egy-egy helyiség leírására használt szöveg hossza maximum 240 karakter, és a tárgyak leírására használt rövid szöveg is rendszerint legalább egy sor hosszú (64 karakter). Ezért a 4-es sorban egy CLEAR 500 utasítás áll. Ez jó 500 karakternyi munkaterületet tart fenn a *Kardhalak és kincsek*-ben használt néhány karakteres változónak. A CLEAR 500 ezenkívül alapállapotba hozza az összes változót, amit a program inicializálásának korai szakaszában tanácsos megtenni.

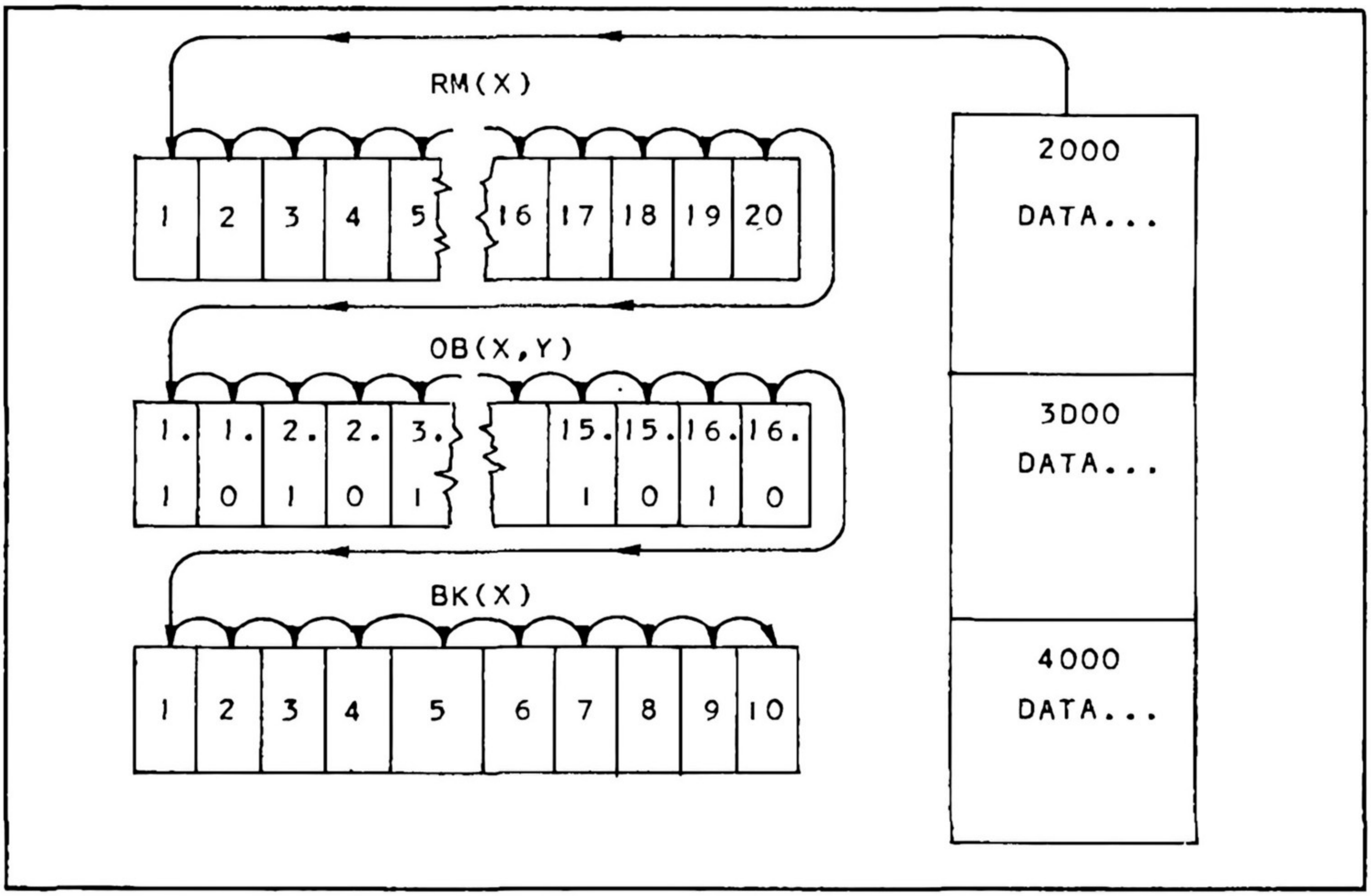
Az előző fejezetben már említettük, hogy a numerikus változókat egésznek deklaráljuk, annak a nemes célnak az érdekében, hogy byte-okat takarítsunk meg.

*A HT-1080Z-nél ez nem így van, nem kell ügyelni a páratlan számra. (*A fordító*)

A DEFINT A–Z tudatja a géppel, hogy minden numerikus változó, amely A és Z közötti betűvel kezdődik (lényegében az összes ilyen változó) egészként kezelendő.

Úgyszintén, az összes tömbszerkezetűnek választott változót megfelelő méretűre kell beállítani, vagyis „dimenionálni kell". A gépen az összes tömb minden indexe tizenegy értéket vehet fel (0-tól 10-ig), hacsak a program kifejezetten másképp nem deklarálja. Más szóval, ha valamelyik sorban A(3)-ra hivatkozunk, a BASIC létrehoz egy A(*n*) tömböt, ahol *n* 0-tól 10-ig terjedő értéket vehet fel. Ha a tömbnek csak néhány elemét akarjuk használni, a többi kárba veszett tárat jelent. Másrészt, ha megpróbálunk pl. A(22)-re hivatkozni, indexhatár-túllépési hiba lesz az eredmény; túlléptük a tömb előre meghatározott méretét.

Alkalmazzuk a DIM utasítást, hogy a kisméretű tömbűknél helyet takarítsunk meg, és azért, hogy a nagyobb méretű tömbök használatát lehetővé tegyük! Figyeljük meg, hogy a 4-1. ábrán egyetlen DIM utasítással deklarálunk öt különböző tömböt! Először a TX$(*n*) karakterláncot deklaráljuk, ezután a DA(*n*) adatok elérésére használt tömböt, majd az RM(*n*) helyiségek állapottömbjét, végül az OB(*m*, *n*) tárgyak állapottömbjét és a BK(10) akadálylistát.



4-2. ábra. Így kapnak kezdőértéket a fontosabb indexes változók az első három DATA blokkból

A 4-es sor hátralevő része ezekből hármat inicializál. Talán még emlékszik rá az Olvasó, hogy a program első három adatblokkja a helyiségek, tárgyak és akadályok állapotának beállítására való. A 4-2. ábra azt mutatja, hogy a megfelelő tömbök hogyan töltődnek föl a három blokkból. Mivel az adatok olvasását irányító mutató a BASIC tároló legelejére áll vissza, indításkor az első READ utasítás az első DATA utasításhoz fér hozzá, majd a rá következő READ utasítások rendre veszik a soron következő blokkokat. A kezdeti beállítás után azonban az adatokhoz az előző fejezetekben leírt módon férünk hozzá, vagyis az adatok olvasását POKE-kal irányítjuk a programban, tehát elsősorban nem a BASIC kezeli.

A 6-os és 8-as sorban végrehajtott inicializálást már leírtuk az előző fejezetben. Ennek a két sornak a végrehajtása után az adatok elérését biztosító DA(*n*) tömb tartalmazni fogja mind az öt fontos adatblokk tárbeli kezdőcímét.

A 10-es sor a játék elkezdéséhez szükséges előkészületek utolsó mozzanata. A CT(0) indexes változó tartalmazza azt a helyiségszámot, ahol a rettenthetetlen kalandozó éppen tartózkodik. A játékos az 1-es helyiségben, a támaszponton kezd, ezért CT(0)-ba 1 kerül. Ezután azt a számlálót kel inicializálni, amelyik a harcias lény, jelen esetben a Kardhal megjelenését irányítja. CT(12)-be egy 10 és 20 közötti véletlen értékeket teszünk az RND függvény segítségével. Végül letöröljük a képernyőt, levesszük a játék címét, és felkészítjük a képernyőt a következő helyiség leírására.

Leírás

Talán emlékszik rá az Olvasó, a 100 és 199 közötti sorokat a Vezérlő foglalja el; ez a programnak az a része, amely nagyon igénybe van véve, mert a legtöbb kezelő a Vezérlőbe tér vissza. A 4-3. ábrán látható a teljes Vezérlő. Az első két sor, a 100-as és a 102-es, a Vezérlő leírás alárendelt része. Ezek rajzolják meg a kalandozót körülvevő közvetlen környezetet. A 104-es sor egy olyan programrészre adja a vezérlést, amelyik a harcias Kardhal tevékenységét irányítja. A maradék sorok, 105-től 110-ig, alkotják a parancsalrészt. Ezek a sorok fogadják a billentyűkről bejövő üzenetet, elemzik a parancsot, és a vezérlést a megfelelő kezelőnek adják át. Először vessünk egy pillantást a leírás alárendelt részére!

Ennek az alárendelt résznek három leíró feladatot kell ellátnia:

- le kell írnia magát a helyiséget,
- le kell írnia a helyiségben levő tárgyakat, és
- le kell írnia a helyiségben esetleg ott levő ellenfelet.

Nézzük először magának a helyiségnek a leírását! A helyiséget — mint jól tudjuk — kétféleképpen írhatjuk le: egy bővebb és egy rövidített leírással. Melyik leíró bekezdést, ill, kifejezést kell megjeleníteni? A szabály a következő: ha először kerestük fel a helyiséget, akkor a bővebb bekezdést kell kiírni.

```
100 CT(5)=RM(CT(0)):GOSUB1000:C=CT(6):GOSUB
1160:GOSUB1180:IFB=0ANDC=0THENCT(6)=1:GOSUB
1020:RM(CT(0))=CT(5):ELSEIFB=1THENN=RND(100
):IFN<20THENB=5:GOSUB1100:GOTO580
102 GOSUB1140
104 GOTO112
105 INPUTA$
106 GOSUB1060:A$=TX$(2):GOSUB1080
108 CT(5)=N:GOSUB1000:IFCT(10)=0ORN=0THENB=
7:GOSUB1100:GOTO104
110 ONCT(6)+CT(7)*10GOTO200,220,240,260,280
,300,320,340,360,380,400,420,460,480,500,52
0,540,560,580,600,620,640,660,680,700
```

4-3. ábra. A Vezérlő, a leíró és a parancs alárendelt résszel

Ezt követően a rövid leíró kifejezést. Először tehát azt az információt kell ellenőrizni, hogy jártunk-e már ebben a helyiségben előzőleg, vagy sem.

Az RM(*n*) helyiségek állapotvektora tartalmazza ezt az információt. Ha az RM(*n*)-ben tárolt egész első számjegye nulla, akkor a helyiségben még nem jártunk; ha egy, akkor már jártunk ott legalább egyszer. Ellenőriznünk kell ezt a számjegyet, tehát nagyon jól jön az 1000-es sorban található Analyz szubrutin. Az Analyz öt számjegyre bontja fel azt az egészet, amelyet ideiglenesen CT(5)-be teszünk, a számjegyeket CT(6) és CT(10) közti változókba rakja. Analyz használata után CT(6)-ban lesz az első számjegy, amelyik megmondja, hogy melyik leírást kell használni.

A 100-as sor azzal kezdődik, hogy CT(5)-be teszi a jelenlegi helyiség állapotát jelző egészet, aztán meghívja Analyz-t. (Figyeljük meg, hogy CT(0)-ban van a jelenlegi helyiség száma, ezért RM(CT(0)) adja meg a szükséges állapotjelző egészet.) Az Analyz befejezésekor CT(6) értéke vagy nulla, vagy egy, attól függően, hogy jártunk-e már a helyiségben.

Ez az elrendezés kényelmesnek bizonyul. A helyiség leírását ténylegesen kiíró szubrutin (neve Viewrm) aszerint, hogy a C változó értéke nulla, a hosszú formát, egyébként a rövid formát használja.

CT(6) már teljesíti ezt a követelményt (biztosíthatom az Olvasót, hogy ez nem véletlen). Nem kell mást tenni, mint C-t egyenlővé tenni CT(6)-tal, és Viewrm-et meghívni, és a megfelelő leírás megjelenik a képernyőn.

Nyilvánvaló, hogy alaposan oda kell figyelni arra, hogyan lesz egy helyiség leírva, ezért egy kis kitérőt teszünk a Vezérlő tárgyalása közben. A 4-4. ábrán

látható Viewrm kódja és egy másik hasznos szubrutiné, Darkck-é. Ne aggódjunk, mindjárt fény derül kapcsolatukra!

Mielőtt Viewrm leírhatna egy helyiséget, függetlenül a hosszú vagy rövid leírástól, figyelembe kell venni még egy utolsó szempontot — nincs-e túl sötét ahhoz, hogy látni lehessen? Emlékezzünk rá, hogy a *Kardhalak és kincsek*-ben (és egy sor más hasonló kalandprogramban) a cselekmény jórészt a föld felszíne alatt játszódik, homályos barlangokban. Ilyen esetekben a szokásos felszereléshez egy fáklya vagy lámpás tartozik — hogy lássunk (programunkban ez a 9-es tárgy). Azaz két kérdésre kell válaszolni: a kalandozó sötét helyiségben van-e? Van a kalandozónál fáklya?

A Darkck szubrutin (DARK Check*-ből ered a neve) ezt a két kérdést értékeli, ezért hívja meg Viewrm Darkck-t mielőtt bármi mást tenne. Nézzük az 1180-as sort! A következő mondatban leírt logikát követve Darkck egyet tesz a B változóba, ha a kalandozó nem láthatja a környezetét. Ha a játékosnak nincs fáklyája, és nincs a föld színe fölött, akkor túl sötét van ahhoz, hogy lásson. Az OB(9,1) indexes változó mondja meg, hol van a fáklya. Ha a kalandozó viszi magával, akkor OB(9,1)-nek 21-gyel kell egyenlőnek lennie, ez a hátizsákot jelölő szám. Ezen túl csak az 1-es és 2-es helyiség van a föld felszínén, ahol nem kell külön fény. Ha CT(0), a játékos jelenlegi helye nem 1 vagy 2, akkor szükség van fáklyára. Darkck ezeket az összehasonlításokat végzi el, és ezeknek megfelelően állítja be B-t.

Visszatérve Viewrm-re, Darkck tehát meg lett hiva. Ha B egyenlő 1-gyel, túl sötét van a helyiség leírásához. Ebben az esetben a „TÚL SÖTÉT VAN... VEREMBE ESHET!" üzenet jelenik meg a leírás helyett. Ez a 39-es üzenet; nem kell mást tenni, mint B-ben beállítani ezt az üzenetszámot és meghívni Mesprt-t (message-print)**az 1100-as sorban. Megjelenik az üzenet és Viewrm visszatér. (Ismétlésként ellenőrizzük Mesprt működését az előző fejezetből.)

Azonban, ha a kalandozó láthat, Viewrm tovább halad. A hosszú és rövid leírás a helyiségeket leíró adatblokkban van. Access segítségével (1040 és 1042 közötti sorok) a megfelelő hosszú bekezdésnyi és rövid kifejezésnyi leírás kiolvasható a DATA sorból és két külön szöveges változóban tárolható. A hosszú változat TX$(0)-ban, a rövidebb TX$(1)-ben van tárolva.

Emlékezzünk rá, hogy az Access-nek két fontos információ kell: a blokk száma az A változóban, és az elem száma a B változóban. Helyiségeknél a blokk száma 5, az elem száma megegyezik a CT(0)-ban tárolt jelenlegi helyiség számával. Tehát Viewrm beállítja e két változót és meghívja Access-t. Mikor Access lefut, a BASIC adatmutatója a két leírást tartalmazó sor elején áll. Beolvasásuk TX$(0)-ba és TX$(1)-be egyszerűen megy.

*Sötétség-ellenőrző. (*A fordító*)

**Üzenet nyomtatása (*A fordító*)

```
NÉV:               VIEWRM
TIPUS:             SZUBRUTIN
BEMENŐ ADAT:       C=0 bövebb leirásnál
                   C=1 rövid leirásnál
EREDMÉNY:          Megjelenik a helyiség
                   leirása, ha látni.
                   Egyébként figyelmezetetö
                   üzenetet ir ki


1160 GOSUB1180:IFB=1THENB=39:GOSUB1100:RETU
RN:ELSEA=5:B=CT(0):GOSUB1040:READTX$(0),TX$
(1):IFC=0THENPRINTTX$(0):RETURN:ELSEPRINTTX
$(1):RETURN


NÉV:               DARKCK
TIPUS:             SZUBRUTIN
BEMENŐ ADAT:       nincs
EREDMÉNY:          B=1 ha tul sotét van
                   B=0 egyébként


1180 IFOB(9,1)<>21ANDCT(0)<>1ANDCT(0)<>2THE
NB=1ELSEB=0
1182 RETURN
```

4-4. ábra. A Wiewrm és a Darkck szubrutin

Az utolsó megfontolandó dolog az, hogy melyik leírást használjuk. Annak idején azonban a Vezérlőben beállítottuk a C változót, hogy a megfelelő leírást válassza ki! Viewrm egyszerűen ellenőrzi C értékét, és TX$(0)-t vagy TX$(1)-et írja ki. Pofonegyszerű!

Átismételve mit is láttunk: a Vezérlőnek le kell írnia a helyiséget. Meghívja Viewrm-et, amelyik vagy a hosszú leírást írja ki, vagy a rövidet — vagy úgy dönt, hogy nem ír ki semmit, mert a helyiség túl sötét.

A helyiség tekintetében még egy dologra kell figyelni! Most, hogy jártunk a helyiségben, meg kell változtatni az állapotvektor elemét, hogy ezt a tényt tükrözze. Ennek az elemnek a számjegyei még mindig a CT(6) és CT(10) közötti változókban vannak. Egyszerűen megváltoztathatjuk CT(6)-ot 1-re. Aztán meghívjuk a Synthe szubrutint, amely összerakja a számjegyeket és egy teljes egészként CT(5)-be rakja. (Synthe, amelyet az előző fejezetben ismertettünk, az Analyz ellentettje.)

Nem lenne helyes, ha ezt a változtatást abban az esetben is végrehajtanánk, ha a helyiség sötét volt, és nem írtunk ki leírást. Miért? Mert ha a kalandozó később visszatér fáklyával a kezében, csak a rövid leírást kapja; már járt ott akkor is, ha semmit sem látott. Ez nagyon sportszerűtlen lenne (és a kalandozónak minden elérhető segítségre szüksége van). Ezért mielőtt „bejárt"-ra változtatjuk a helyiség állapotát, kérdezzünk rá, látott-e valamit? Ez azt jelenti, hogy ismét meg kell hívni Darkck-t, amelyik megfelelően beállítja B-t.

Ha B nulla, és C (ezt jó régen beállítottuk a helyiség állapotára) is nulla, akkor a helyiséget bejártnak tekinthetjük. Ebben az esetben állítsuk be CT(6)-ot 1-re, és hívjuk meg Synthe-t (1020-as sor)! CT(5) most a helyiség új állapota, ezt betehetjük RM(CT(0))-ba.

Mi történjen, ha a helyiség sötét? Ebben az esetben játsszunk egy kicsit szegény kalandozóval! Emlékszik az Olvasó a „TÚL SÖTÉT VAN ... VEREMBE ESHET" üzenetre? Hát, adjunk egy esélyt neki! A BASIC RND függvényével állítsunk elő egy 0 és 100 közötti véletlenszámot, adjunk a játékosnak 20 százalék esélyt arra, hogy verembe esik és az eséstől meghal. Az N változóba kerül a véletlenszám; ha N 20-nál kisebb, sorsa megpecsételődött. Mesprt segítségével kiíratjuk az 5-ös üzenetet („HALÁLRA ZÚZTA MAGÁT..."), és a program elugrik arra a kezelőre, amelyik a halott kalandozóval foglalkozik. Talán kegyetlennek és sportszerűtlennek tűnik, de egyszerűen csak egy eszköz abból a célból, hogy visszatartsuk a beképzelt játékosokat, hogy megkíséreljék a teljes helyszín bejárását fáklya segítsége nélkül.

A tárgyak számontartása

Most, hogy a helyiséget leírtuk, a tárgyak következnek. Ügyeljünk arra, hogy ebbe beletartoznak a passzív lények, amelyek nem támadnak, amíg a kalandozó nem ingerli őket! Ezt a követelményt a végrehajtó ismét egy másik szubrutin segítségével oldja meg. A végrehajtó 102-es sora hívja meg.

A 4-5. ábrán látható az 1140-es sor, a Listob szubrutin (neve a „list-objects"*-ből származik). Feladata, hogy a tárgyak állapotmátrixát teljesen átvizsgálja, kikeresse a jelenlegi helyiség tárgyait és kiírja leírásukat.

Most már valószínűleg nem lepődik meg az Olvasó az első néhány utasításon. Ismét csak az az értelme, hogy ha túl nagy a sötét, a tárgyak leírása nem írható ki! Ismét itt tartunk..., újra meghívjuk Darkck-t és ellenőrizzük, hogyan lett a B változó beállítva. Listob szó nélkül tér vissza, ha a környezet túl sötét.

*Tárgyak felsorolása. (*A fordító*)

```
NÉV:             LISTOB
TÍPUS:           SZUBRUTIN
BEMENŐ ADAT:     nincs
EREDMÉNY:        felsorolja a helyiségben
                 található összes tárgyat
                 ha elég világos van

1140 GOSUB1180:IFB=1THENRETURN:ELSEA=4:FORB
 =1TO16:IFCT(0)<>OB(B,1)THENNEXTB:RETURN:ELS
EGOSUB1040:READTX$(4):PRINTTX$(4):NEXTB:RET
URN
```

4-5. ábra. A Listob szubrutin

Normális esetben azonban a tárgyak láthatók, és Listob hozzálát a leírásukhoz. A tárgyak leírása a 4-es számú adatblokkban van, innen Access segítségével hívjuk meg. Az A változót 4-re állítjuk be, arra számítva, hogy Access-t többször is meghívjuk. Az Access másik szükséges változóját, B-t a következő ciklus állítja be.

A tárgyak állapotmátrixában OB(n,1) adja meg annak a helyiségnek a számát, ahol a tárgy található. Mely tárgyak vannak a jelenlegi helyiségben? Egy 16-szor lefutó FOR-NEXT ciklust írunk, mivel 16 tárgy van. Ha a tárgy nincs a CT(0)-lal jelzett helyiségben, akkor kihagyjuk, egyébként pedig kiíratjuk.

Ilyen esetben meghívjuk Access-t, hogy a leírást megkapjuk. Az A változó már be van állítva, hogy a megfelelő adatblokkot határozza meg. Az Access-nek szüksége van még a B változóra, hogy tudja, a blokk melyik elemére mutasson.

Szerencsére gondunk volt rá, hogy B-t használjuk a FOR-NEXT ciklusban. Így B már egyenlő a keresett tárgy számával, és az Access-nek minden rendelkezésére áll, amire szüksége van, hogy megkeresse a kiírandó mondatot. Amint Access lefutott, Listob egy szokásos BASIC-beli READ utasítással hozzáfér a leíráshoz. A mondat TX$(4)-ben tárolódik és azonnal kiírásra kerül. Az 1140-es sor hátralevő része a többi tárgy ellenőrzésével teszi teljessé a ciklust.

Az eddigiekben a Vezérlőleíró alárendelt része tájékoztatott a helyiségről, és felsorolta a köröskörül heverő tárgyakat — többek között a „szunnyadó" lényeket is. De mi legyen a félelmetes, kitartó Kardhallal?

A kóborló szörnyek

A Vezérlő utolsó leíró jellegű feladata, hogy a szívós, harcias *Kardhal* jelenlétével és támadásával riogassa a kalandozót. Ez a lény nem pusztán meggátolja, hogy átjussunk egy megadott kapun, hanem amint nevéből is következik, sohasem adja fel. (Ha egyszer rád talál, helyiségről helyiségre követ, míg vagy Te, vagy ő ki nem múlik!) Kiszámíthatatlanul támad, és épp ennyire kiszámíthatatlan, hogy a rendíthetetlen játékosnak sikerül-e megölnie. Az a BASIC kódrész, amelyik ezt a lényt irányítja, a Vezérlőben található.

A 4-6. ábrán a harcias Kardhallal foglalkozó programrész látható. Három változó szabályozza a gonosz bestia megjelenését és tevékenységét. Az OB(0,1) indexes változó, amely a tárgyak állapotmátrixának egy fel nem használt eleme, tárolja a helyiség számát, ahol a Kardhal található. Másrészt OB(0,0) egy jelző. Ha értéke nulla, a Kardhal még nem akadt rá a kalandozóra. Ha értéke egy, akkor a Kardhal és a játékos ugyanabban a helyiségben van. Végül CT(12) számláló, azt ellenőrzi, milyen gyakran botlik bele a hős a Kardhalba.

Hogyan talál a Kardhal a játékosra? Ez sokféle módon megoldható. Pl. volt egy olyan változat, amelyikben egy véletlenszám-generátor juttatta a Kardhalat egyik helyiségből a másikba, amíg rátalált a játékosra. Ezzel a megközelítéssel — és sok más hasonlóval — az a baj, hogy túl véletlenszerű volt. Néhány menetben a Kardhal sosem jelenik meg; más esetekben pedig minden második lépésnél felbukkan! Világos, hogy gátat kell szabni véletlenszerű vándorlásának.

Ebben a változatban CT(12) egy számláló, amely egy 10 és 20 közé eső véletlenszámmal van feltöltve. A számláló eggyel csökken a játékos minden lépésénél. Mikor elfogy, a játékos találkozik a Kardhallal! Az Olvasó megváltoztathatja a találkozás gyakoriságát, de maga az ötlet bevált.

```
112 IFOB(0,0)=0ANDCT(0)>2THENCT(12)=CT(12)-
1:IFCT(12)<=0THENCT(12)=RND(10)+10:OB(0,1)=
CT(0):OB(0,0)=1:GOTO116:ELSE105
114 IFCT(0)<>3THENOB(0,0)=0:GOTO105:ELSEOB(0
,1)=CT(0)
116 B=42:GOSUB1100:B=RND(100):IFB<75THEN125
ELSEB=43:GOSUB1100:B=RND(100):IFB<60THENB=4
4:GOSUB1100:GOTO580:ELSE105
```

4-6. ábra. A kitartó Kardhalat vezérlő programrész

Nézzük a programrészt! Első feladat a CT(12) számláló csökkentése. A számlálót csak két esetben szabad csökkenteni. Először a játékos és a Kardhal még nem lehetnek együtt, mivel a számlálóval pont erre készülünk. Másodszor a játékos nem lehet az 1-es vagy 2-es helyiségben, mivel ezek a föld felszínén vannak és a Kardhalak gyűlölik a szabad levegőt! Az OB(0,0) jelző biztosítja az első feltételt, CT(0) a másikat. Ha a játékos a föld felszínén van, vagy a Kardhal vele van, a sor hátralevő részét kihagyjuk. Egyébként CT(12) eggyel csökken.

De mi történjen, ha CT(12) végül nullára csökken? Akkor megjelenik a Kardhal! Először CT(12) ismét beáll valamilyen szintre, hogy szabályozza a következőként megjelenő Kardhalat. Másodszor a Kardhal helye egyenlő lesz CT(0)-lal, a hős helyével. (Aztán a 116-os sorban a Kardhal eldönti, hogy támadjon-e vagy sem.) Ha CT(12) még nem fogyott ki, a programrész egyelőre befejeződik és visszatér a program beolvasó részébe.

Ha a játékos a föld színén van, vagy a Kardhal vele van, végrehajtódik a 114-es sor. Az első esetben a Kardhal magára hagyja a játékost, ha a játékos a föld felszínén mozog. Azután OB(0,0)-ba nulla kerül, jelezve, hogy a Kardhal már nincs többé a hős nyomában. (Ez ismét elindítja a CT(12) visszaszámlálását egy következő találkozás céljából.) A másik esetben a Kardhal követi a játékost; így a Kardhal helyzetét jelző szám OB(0,1)-ben egyenlő lesz a játékoséval CT(0)-ban. A 116-os sor vezérli a Kardhal esetleges támadását, három lehetőséget állítva elő: a Kardhal nem támad, a Kardhal támad, de nem öl, a Kardhal támad és megöli a kalandozót.

Mielőtt a három lehetőséggel bűvészkednénk, a 42-es számú figyelmeztető üzenet jelenik meg: „EGY DÜHÖS KARDHAL VAN A KÖZELBEN!" Előállítunk egy 0 és 100 közé eső véletlenszámot. Ha ez a szám 75-nél nagyobb, az első lehetőség teljesül: a Kardhal nem támad, a program folytatódik.

Ha a Kardhal támad, aminek 75 százalék az esélye, a 43-as üzenet felkiált: „ELŐRE RONT EGY FEKETE GÖRBE KARDDAL!" A párbaj kimenetelét ezután egy második véletlenszám határozza meg. Egy 60-nál nagyobb érték hősünk halálát jelenti. Ebben az esetben a 44-es üzenet elpanaszolja: „A KARDHAL MISZLIKBE APRÍTJA". Ezután a program elugrik egy olyan programrészre, amely ügyesen feltámaszt és újra beléptet a helyszínre. Egyébként a program a beolvasórésznél folytatódik. Nem hinnénk, hogy a megfelelő képzelőerővel rendelkező programozó Olvasóknak magyarázni kellene, hogy ezek önkényes valószínűségek. A 116-os sorban található számok megválasztása bizonyítja a játékos iránti szánalmunkat és kegyetlenségünket.

Ha ezzel megvagyunk, a Vezérlő kiírja a képernyőre az üzenetet, és türelmetlenül várja a játékos üzenetét, aki kétségkívül szeretné nekiszegezni kardját a Kardhalnak, mielőtt a százalékok „visszalőnek". Most lép színre a Vezérlőparancs alárendelt része.

Parancsra várva

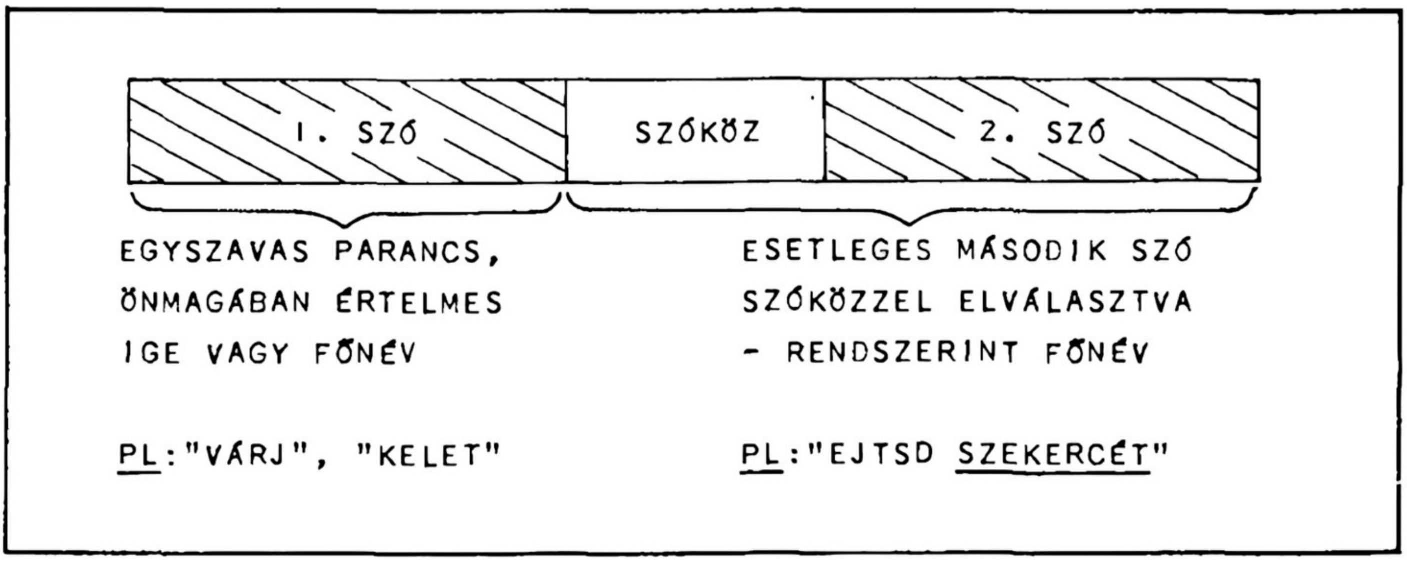
A 105-ös és 110-es sor között található a parancsokkal foglalkozó alárendelt rész. Ez végzi a játékos egy-, kétszavas kifejezéseinek részekre bontását, elemzését, majd végrehajtja a szükséges teendőket.

Igaz, ami igaz, a parancsok elemzése (értelmezése) tekintetében vannak sokkal elegánsabb kalandprogramok is! Néhány közülük megenged elöljárós kifejezéseket, határozókat, ill. hasonlókat. Ezek remek apróságok, ha rendelkezésünkre áll mind a tár, mind a végrehajtási sebesség, amely egy nagyméretű szótár és egy sor más lehetőség gyors kezeléséhez szükséges. Lehetőségeinket megköti a BASIC és a 16K. Ne csüggedjünk: a kétszavas parancsok is megteszik, ha a szótárt ügyesen választjuk meg!

A 4-7. ábrán látható a kiválasztott nyelvtan. A billentyűkről bejövő parancsok egy-, kétszavasak: egy magában álló ige vagy főnév (ÉSZAK, ill. NYISD), olykor pedig egy ige és egy főnév (VEDD FEL A GYÉMÁNTOT, ill. MENJ NYUGATNAK).*

Az 1. szóra hárul, hogy meghatározza a kért feladat típusát, oly módon, hogy egy megfelelő szubrutint vagy kezelőt meg lehessen hívni. Pl. amikor a játékos azt kéri, hogy „PONTOZZ", működésbe lép egy kezelő, amely kijelzi a jelenlegi pontszámokat, azután visszatér a Vezérlőbe.

A 2. szó megadja az első szóból következő feladat paramétereit. Tegyük fel, azt parancsoltuk, hogy „VEDD"! Mit csinál a program? Meghívja a Vedd nevű kezelőt. Igen ám, de a helyiségben levő tárgyak közül melyiket akarja a kalandozó felvenni? A második szó megszünteti a kétértelműséget, mert további adatot szolgáltat.



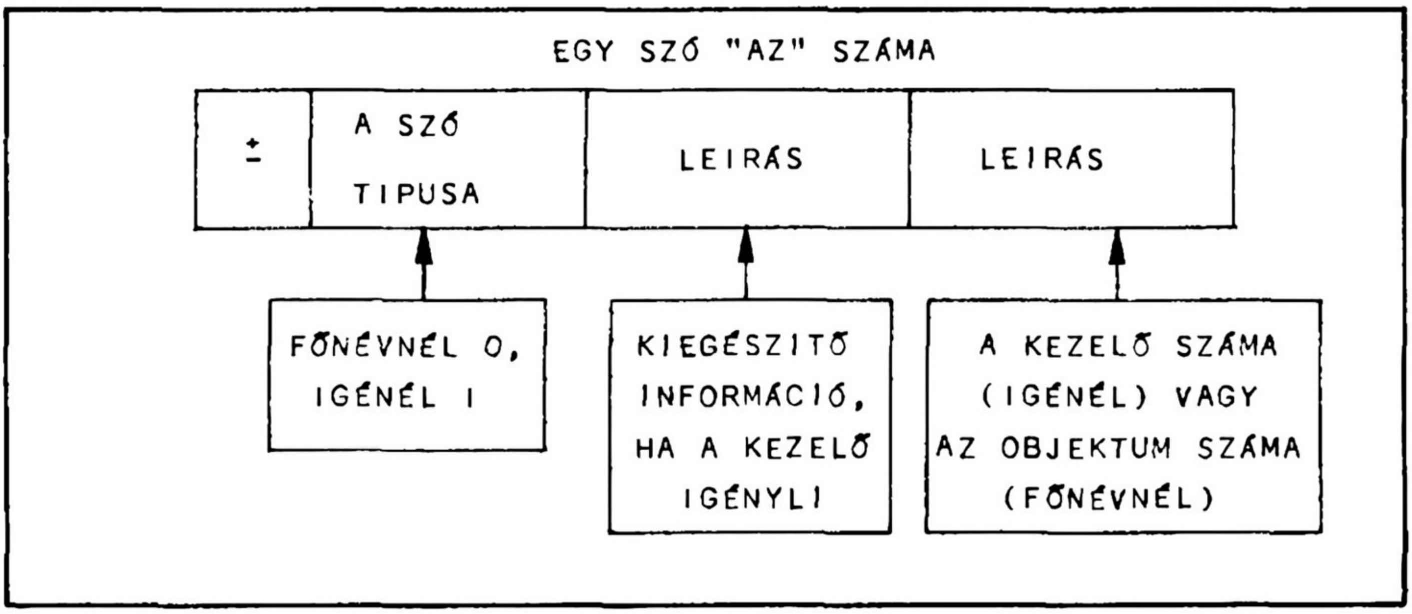
4-7. abra. A KARDHALAK ES KINCSEK leegyszerusitett nyelvtana

*Az angol és a magyar nyelvtan nagyon eltér. Az angolban ezek ragozatlan szavak. Elemzésük magyarul jóval nehezebb, a pontos különbségek megtalálhatók a könyv elején, *Az előszó a magyar kiadáshoz* c. részben. (*A fordító*)

Mindenesetre, a parancs szavait azonosítani kell. Ez bizony a programozó-szakma rettegett fogásával jár együtt — a *táblázatokban való kereséssel.* A tárban fenn kell tartanunk egy szótáblázatot, hogy minden bejövő szót összehasonlíthassunk a táblázat szavaival.

A *Kardhalak és kincsek* szótáblázatát a 2-es adatblokk alkotja. Természetesen nem elegendő, hogy egy hosszú, szavakból álló listánk legyen; minden szóhoz tartoznia kell még egy adatnak is, amely tájékoztatja a parancsértelmezőt, hogy miként értelmezze. A táblázatban szereplő minden szót összekapcsolunk egy szóazonosító AZ nevű egésszel. Ennek az egésznek minden számjegye a vele összekapcsolódó szó értelmezéséről tartalmaz információt.

A 4-8. ábrán látható a szóazonosító lebontása. Három információs mező segít egy megadott szó azonosításában. Az első az 5. számjegy; ha értéke egy, akkor a szó jogosan szerepel első szóként, és ennek megfelelően kell értelmezni. Minden olyan szó, amelynek AZ szám 10000 és 19999 között van, meghív egy kezelőt, de melyiket? A választ az 1. és 2. számjegyből álló mező adja meg. Ez választja ki a 99 lehetséges kezelő egyikét, amely ezzel a szóval járhat. Pl. a „PONTOZZ" a szótáblázatában a 10012-es AZ-vel található. Ebből a parancsértelmező már tudja, hogy a 12-es kezelőt kell behívni.



4-8. abra. Az egyes szamjegyek jelentese az AZ szamban

Ide tartozik egy harmadik mező is, ezt a 3. és 4. számjegy alkotja. Néhány különleges kezelő számára tartogat többletinformációt. Használatára a legegyszerűbb példa a Liners nevű kezelő. Ez a kezelő egyszerűen egysoros választ ad az egyszavas bemenő üzenetre. Ha a játékos azt írja: „VÁRJ", akkor a „MÚLIK AZ IDŐ" választ adja. Több különböző szó hívhatja be egyébként a Liners-t, de melyik üzenetet írja ki? Egyszerűsíti a dolgunkat, ha a szó AZ számának harmadik mezeje annak az üzenetnek a számát tartalmazza, amelyet Liners használ az adott szóval kapcsolatban. A szótáblázatban a VÁRJ a 13809

egésszel kötődik össze. Ez tudatja az értelmezővel, hogy a 9-es kezelőt hívja be (magát a Liners-t, és tudatja, hogy a 38-as üzenetet írja ki, ami nem más, mint a „MÚLIK AZ IDŐ" üzenet). Más kezelők is használják ezt a harmadik mezőt, de a dolog lényege ebből már érthető.

Térjünk vissza az első mezőre! Ha értéke egy, van egy érvényes első szavunk; ha értéke nulla, akkor ez egy érvényes második szó. A kezelőnek szüksége van némi járulékos információra, mint pl. hogy melyik tárgyat VEGYE fel, vagy melyik lényt ÖLJE meg. Ha a második szó feladata az azonosítás, a második mező (első és második számjegy) jelenti annak a tárgynak a számát, amelyikre a szó vonatkozik. (A harmadik mező kihasználatlan.) Lényegében a szótáblázatban a tárgyak neve a tárgy számához kapcsolódik, mivel az AZ szám többi számjegye nulla.

Azt is figyeljük meg, hogy a szótáblázat számos különböző szava ugyanarra a tárgyra vonatkozik: azonos AZ számokkal kell őket párosítani. Pl. mind az ÉKKŐ, mind a KORONA szó 1-es AZ számmal kapcsolódik össze, mert mindkettő az ékköves koronára, az 1. tárgyra vonatkozik.

Itt az ideje, hogy megnézzük magát a kódot, amely felhasználja a szótáblázatot, a szó AZ számát, és meglássuk, hogyan hívjuk be a kezelőket! Okoskodásunkban a 4-3. ábrára hivatkozunk.

Az első lépés egyszerű — olvassuk be a szöveget! Az INPUT A$ utasítás kiír egy kérdőjelet a képernyőn, és addig vár, míg a játékos egy sorozatkaraktert be nem ír, amit ENTER zár le. A bejövő szöveg az A$ változóba kerül.

Most gondolkozzunk el egy pillanatra! Hol van az első és hol a második szó abban a bizonyos A$ változóban? A gép számára A$ csak egy karaktersorozat! Léteznie kell olyan eljárásnak, amelyik felbontja A$-t szavakra.

Az eljárás bele van ágyazva a GETCOM nevű szubrutinba, ami a 4-9. ábrán az 1060-as sorban található.

A GETCOM célja, hogy az A$ karakterláncban elkülönítse a szavakat és a TX$(2), TX$(3) változóba tegye, első, ill. második szóként. Ha csak egy szó van, az TX$(2)-be kerül, mint első szó, és TX$(3)-ba nulla hosszúságú szöveg kerül, jelezvén, hogy nincs második szó.

A szétválasztási folyamat kulcsa a szóközkarakter. Ha az A$-ban tárolt bejövő lánc tartalmaz egy szóközt, feltételezzük, hogy ez az elválasztó az első és a második szó között. Ha nincs szóköz, A$-t úgy tekintjük, hogy teljes egészében az első szó. Getcom-nak módszeresen meg kell vizsgálnia A$-t, szóköz után kutatva.

Szerencsére a Microsoft BASIC tartalmaz néhány nagyon hasznos szövegkezelő függvényt. A LEN(X$) függvény meghatározza a bemenő szöveg hosszát, így tudjuk, meddig keressük.

A MID$(X$,y,z) függvény segítségével pedig bizonyos karaktereket elkülöníthetünk a karakterláncból.

Lássuk, hogyan is csináljuk! Írunk egy ciklust, amelyik az elsőtől az utolsó karakterig átvizsgálja a láncot. A lánc minden karakterét összehasonlítjuk a

```
NÉV:              GETCOM
TIPUS:            SZUBRUTIN
BEMENO ADAT:      A$=a leirt parancs
EREDMÉNY:         TX$(2)·az 1. szo
                  TX$(3)=az utolso szo

1060 FORI=1TOLEN(A$):IFMID$(A$,I,1)<>" "THE
NNEXTI:TX$(3)="":TX$(2)=A$:RETURN:ELSETX$(2
)=LEFT$(A$,I-1):FORI=LEN(A$)TO1STEP-1:IFMID
$(A$,I,1)<>" "THENNEXTIELSETX$(3)=MID$(A$,I
+1):RETURN
```

4-9. ábra. A Getcom szubrutin

szóközzel. A MID$(A$,I,1) kifejezés egy karaktert választ ki A$-ból, mégpedig azt, amelyik I karakternyire van a lánc elejétől. Miközben I értéke változik, minden egyes karaktert ellenőrzünk, vajon szóköz-e? Valahányszor úgy találjuk, hogy egy karakter nem szóköz, a ciklus folytatódik.

Ha a ciklus lefut anélkül, hogy szóközt talált volna, TX$(3) hossza nullára lesz beállítva, ami azt jelenti, hogy a láncban nincs második szó. A$-t egy szóként értelmezzük, és TX$(2)-ben tároljuk, mint első szót. Getcom befejeződik és visszatér.

Ellenben, ha Getcom szóközt talál, a szóköztől balra eső karaktereket az 1. szóként feltételezi, és a szóköztől jobbra eső krakterek 2. szóként veszi. Először a második szó kerül TX$(3)-ba, mert a MID$(A$,I+1) kifejezés az I+1-edik pozíciótól a bejövő lánc végéig terjedő karaktereket választja le. (Figyeljük meg, hogy ez a szóközt magát nem tartalmazza.)

Ezután az 1. szót tároljuk TX$(2)-ben. A LEFT$(A$,I–1) kifejezés leválasztja a bejövő lánc kezdetétől az I–1-edik karaktert is beleértve az összes karaktert. (A szóköz ismét kimaradt.) Getcom feladata befejeződött; így visszatér.

Jogos a kérdés: Mi van akkor, ha egynél több szóból áll a bejövő szöveg? Nos, gondoljuk meg! Getcom szétválaszt a legelső szóköznél. Nem folytatja a keresést, hogy megnézze, van-e még szóköz vagy szó. Ezért, ha azt irjuk: „ÖLJ PÓKOT GYORSAN", az 1. szó az, hogy „ÖLJ", és a 2. szó „PÓKOT GYORSAN". Azonnal látjuk, hogy a haszontalan harmadik szó nyugodtan elhagyható, amint az értelmező rájön, hogy milyen lényt akar a második szó kifejezni.

*A gyakorlatban nem tanácsos 249 karakternél hosszabb adatcsoportokat írni. (*A fordító*)

```
NÉV:            IDWORD
TIPUS:          SZUBRUTIN
BEMENŐ ADAT:    A$=egy szo
EREDMÁNY:       N=a szo AZ száma ha meg
                találta a szót a szavak
                táblázatában.
                N=0 egyébként

1080 IFLEN(A$)>5THENA$=LEFT$(A$,5)
1082 A=2:B=1:GOSUB1040
1084 READB$,N:IFB$="."ORB$=A$THENRETURNELSE
1084
```

4-10. ábra. Az Idword szubrutin

Térjünk vissza magához a parancsértelmezőhöz! Miután meghívta Getcomot a bejövő lánc szétbontására, rendelkezésére áll egy első szó. A következő tennivaló: megtalálni ezt a szótáblázatban, megszerezni a szó AZ számát, leválasztani a kezelő számát és a kezelőt behívni. Egyszerű!

Egy másik szubrutin teszi egyszerűvé, aminek a neve Idword; kódja a 4-10. ábrán látható. Az értelmező A$-ba teszi az első szót és meghívja Idword-öt. Idword fogja az A$-ban levő szót, és megkeresi a szótáblázatban. Amikor a szót megtalálja, az N változóba beteszi a szóhoz kapcsolt AZ számot.

A folyamat első lépése a 1080-as sorban lezajlik. Alapjában véve ez a sor mindössze annyit tesz, hogy az A$-ban található szó hosszát öt karakterre korlátozza. Ha az Olvasó megnézte a szótáblázatot, elgondolkozhatott azon, vajon miért öt karakter hosszú az összes tárgy és egyéb fogalom neve. Nos, szigorúan a helytakarékosság miatt. Az a tapasztalat, hogy kalandprogramoknál a szavak felismerése szempontjából az optimális hosszúság az öt betű. Bizonyos, hogy négynél kevesebb betű némi kétértelműséget és hibás azonosítást eredményezhet. Így lehetőség nyílik a beírt üzenetek némi rövidítésére is. A játékos leírhatja, hogy „LELTÁ" és a program tudja, hogy leltárt kér. A kétbalkezes játékos, aki belefeledkezett a játék hevébe, szívesen fogad egy kis pihenést!

Ezután Idword felkészül a szótáblázat időrabló átolvasására. A szótáblázat a 2-es adatblokk és Idword ennek az első eleménél akarja kezdeni a keresést. Ennek megfelelően állítja be az A és B változót, és meghívja a mindig ugrásra kész Access-t, hogy állítsa a BASIC DATA mutatóját a táblázat kezdetére. A soron következő READ utasítások a szótáblázat elemeit olvassák.

Idword a jó öreg, régimódi soros keresést használja: a szótáblázat nincs ugyan ábécésorrendbe szedve, de mint kiderül, a szó megtalálásából eredő idővesztesség nem túl nagy; így elkerülhetjük a bináris kereső szubrutinnal való bíbelődését. (A könyv 10. fejezete ennek ellenére módot ad a szótáblázat ábécésorrendbe szedésével a keresés meggyorsítására.)

Megmaradva a soros keresésnél, Idword párosával olvassa az adatokat. B$-ba pakolja a szót, és a hozzá tartozó AZ számot N-be teszi. Ezután összehasonlítást végez. Világos, hogy ha B$ megegyezik a beírt A$ szóval, a feladat véget ért, és Idword visszatér úgy, hogy a megfelelő AZ szám még mindig N-ben van. Történik azonban még egy összehasonlítás, hogy megállapítsuk, nem a „." karaktert olvastuk-e ki a táblázatból. A szótáblázat utolsó eleme ugyanis mindig egy pont a hozzákapcsolt nulla AZ számmal. Más szóval, ha a keresés során Idword egy pontot olvas, tudni fogja, hogy elérte a táblázat végét anélkül, hogy a keresett szót megtalálta volna. Az előzőhöz hasonlóan nem tesz mást, mint visszatér. Az N változó értéke azonban most nulla, ami a keresés sikertelenségére fenntartott AZ szám. Ha az Idword-öt meghívó program (az értelmező) nullaértéket tartalmazó N-t kap vissza, tudni fogja, hogy a beírt szó nincs a szótárában, és ennek megfelelően válaszolhat.

Ismét csak térjünk vissza az értelmezőhöz! Most, hogy megkapta az N változót, az értelmező megkezdheti N felbontását és felhasználását.

Az Olvasó remélhetőleg még emlékszik az Analyz szubrutinra, ami szétválaszt egy megadott egész számot.

Az értelmező CT(5)-be teszi N értékét, és meghívja Analyz-t. Mikor a szubrutin lefut, az AZ szó öt számjegye a CT(6) és CT(10) közötti változókban található.

Most az értelmezőnek néhány döntést kell hoznia. Mi történjék, ha a játékos egyetlen szót írt be, amelyik igazában nem fogadható el első szóként, pl. „PÓK"? Vagy mi legyen, ha a játékos ugyan két szót írt be, de az első nincs megengedve első szóként, mint pl. a „PÓKOT ÖLD MEG" kifejezés esetén? Az értelmező a fentiek közül egyiket sem fogadja el az AZ szó ötödik számjegyének ellenőrzése után. Mert ennek a számjegynek 1-nek kell lennie egy elfogadható első szónál. Ha nem az, az interpreter süketnek tetteti magát: 7-et állít be B-be, és meghívja Mesprt-t. Eredményül megjelenik a kérdés: „MIT MOND?" A vezérlés ismét az INPUT A$ utasításra kerül, lehetőséget adva a játékosnak új parancs beírására.

Ugyanakkor az értelmező ellenőrzi N értékét, hogy meggyőződjön róla, egyáltalán felismerte-e a szót szótáblázatból. Ha N nullával egyenlő, az értelmező ismét tudatlannak tetteti magát, és új parancsot kér a játékostól a 7-es üzenetben szereplő kérdéssel, visszatér a 104-es sorra és az INPUT A$-ra.

Lelki szemeimmel látom, hogy az Olvasó szkeptikus! Azt kérdi, miért ilyen buta az értelmező. Végül lehetne anyi esze, hogy eltekintsen a szavak sorrendjétől a „PÓKOT ÖLD MEG" példában. Még egy buta embernek is világos, hogy mit akart ez a kifejezés jelenteni: miért nem jó akkor egy számítógép

számára? Ez ismét csak ízlés dolga! A Vezérlőparancs alárendelt részét finomítani lehet, hogy okos és értelmes legyen, ha a programozó hajlandó feláldozni némi tárat és futási sebességet.

Ha a program túljutott a parancsbevitelnek eme néhány megszorításán, készen áll arra, hogy meghívjon egy kezelőt. Az elemzett AZ szó első és második számjegye, amelyek most CT(6)-ban és CT(7)-ben vannak, tartalmazzák a kezelő számát. A két különválasztott számjegyből úgy állítjuk ismét elő az eredeti számot, hogy a második számjegyet megszorozzuk tízzel (mivel az eredeti AZ számban ez volt a tízesek helyiértékén) és hozzáadjuk az első számjegyet. Az eredmény tehát egy kezelő száma 1 és 99 között.

Hálásak lehetünk annak, aki elsőnek javasolta, hogy legyen a BASIC-ben kiszámított GOTO. Ez a funkció ON X GOTO A,B,C,...Z alakban, igen egyszerűvé teszi a program vezérlését. Az ON...GOTO utasítást egy sor BASIC sorszám követi; a GOTO utasítás arra a sorra adja a vezérlést, amelynek sorszámát az utasításban szereplő változó választja ki a listából. Ha a változó a kezelő száma, az ON...GOTO egyezteti ezt a számot a programban elfoglalt helyével és ráadja a vezérlést. (Ügyeljünk rá, ha a változó értéke nulla, semmilyen GOTO nem hajtódik végre, hanem a következő utasítás következik. Abban az esetben pedig, amikor a változó értéke meghaladja a listán szereplő sorszámok számát, hibajelzést kapunk.)

Egy időre a program valamelyik kezelő irányítása alá kerül. A kezelő feladatától függően a vezérlés két lehetséges belépési pont egyikén tér vissza a Vezérlőbe. Az első a leíró alárendelt rész. Miután a játékos lépett a helyszínen, látnia kell a helyiséget, ahova jutott. A Vezérlő leírórésze a logikus visszatérési pont. A másik belépési pont a parancs alárendelt rész. Néhány parancs nem igényli, hogy másodszor is leírást adjon a közvetlen környezetről; ilyen parancsok a PONTOZZ, LELTÁR vagy VEDD. Miután ezek a kezelők elvégezték feladatukat, újabb parancsért térnek vissza.

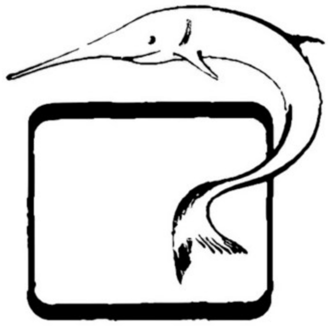
Ez alkotta a kalandprogram magvát. Most röviden megismételjük a program tevékenységét attól a pillanattól kezdve, hogy leírtuk: RUN és ENTER.

1. Inicializálás
- Írd ki a játékos nevét!
- Állítsd be a változókat!
- Töltsd fel a tárgyak állapotmátrixát és az akadálylistát!
- Hozd létre az adatok elérését biztosító vektort!
- Helyezd a játékost az 1-es helyiségbe, töröld a képernyőt és hozd alapállapotba a szívós, harcias lényt!

2. Vezérlés
- Leíró alárendelt rész.
  Írd le a helyiséget!
  Írd le a közeli tárgyakat!
  Írd le a harcias lényt, ha a közelben van!
  Kezeld le a harcias lény támadását!

- Parancs alárendelt rész.
  Olvass be egy parancssort!
  Értékeld ki egy vagy két szó gyanánt!
  Keresd meg az első szót a szótáblázatában!
  Ha lehetséges, hívj meg egy kezelőt a szóból kiindulva!

Az előzőek mindössze berendezték a színpadot egy tényleges *Kardhalak és kincsek* játékra. Lássuk végre, hogyan is játsszák el az egyes kezelők magát a játékot! Logikus annál a kezelőnél elindulni, amelyik a játékost mozgatja. Ez a következő fejezet tárgya.



5. FEJEZET

Bejárjuk a helyszínt

Amint a játékos elkezdte a *Kardhalak és kincsek*-et, egy helyiségben találja magát, és megtudja, hogy mi minden van körülötte. A kezdeményezés a játékos kezében marad. Mit is kellene tennie? A parancsok értelmezője alig várja, hogy a játékos beírjon valamit, és mintegy húsz kezelő áll készen, hogy a játékos parancsát végrehajtsa.

A játékos először rendszerint mozgási parancsot ad ki. (Nyilvánvaló, hogy bővebb képet szeretne kapni a környezetéről.) Ilyenkor jó néhány kezelő lép működésbe.

Mikor egy kaland helyszínét járjuk be, elsősorban háromfajta bejárási parancsot adhatunk ki: explicit bejárási parancsot, implicit bejárási parancsot és mágikus bejárási parancsot.

Az *explicit bejárási parancs* teljes információt ad a bejárás irányáról. Ahogy ezt a 2. fejezetben alaposan megtárgyaltuk, a játékos tíz irányban mozoghat, az iránytű nyolc irányában, valamint fel és le. Az explicit bejárási parancs pontosan közli az értelmezővel a kívánt irányt. A játékos pl. azt írhatja, hogy „MENJ ÉSZAKNAK" vagy egyszerűen „ÉSZAK", sőt esetleg csak annyit, „É". A fenti esetek mindegyikében a parancsértelmező tudja, mit várnak tőle, és megteszi a szükséges lépéseket, hogy a játékos a kívánt irányban mozogjon (feltéve, hogy nincsenek akadályok).

Ettől eltérően, az *implicit bejárási parancs* csak azt jelzi, hogy lépni akarunk; nem adja meg az irányt. Az értelmezőnek kell valamilyen módon kitalálnia a szándékolt irányt, a helyszín alapján. Pl. a játékos egy sziklapárkány szélén áll. Ha azt írja, „UGORJ", ezzel nem mondta meg az irányt — de az értelmező feltételezi, hogy az irány: le.

Hasonlóan, ha egy helyiségnek csak egy kapuja van, észak felé, az értelmező a „KI" parancsot ebben a környezetben azonosan értelmezi a „MENJ ÉSZAKNAK" paranccsal. Az implicit bejárási parancsok több intelligenciát követelnek az értelmezést végző kezelőtől.

A *mágikus bejárási parancsok* a kalandprogramok elengedhetetlen tartozékai és rendszerint kapóra jönnek veszélyes helyzetekben. Ezek a parancsok rendszerint egy vagy több olyan bűvös szón alapulnak, amit a kezelő azonnal megért, és előre beprogramozott mozgást eredményeznek. A mágikus utazás jellegzetesen távolra szállítással jár; ahelyett, hogy egy lépéssel elmozdulna, valamely égtáj felé, a játékos hirtelen egy másik helyiségben találja magát, mely igen messze a kiindulási ponttól. Más tényezőknek is van szerepe (pl. hogyan határozzuk meg a célt), de ezekkel megfelelő helyen fogunk foglalkozni.

A három bejárási parancs kulcsa a hozzájuk kapcsolódó kezelő. Ezért vizsgáljuk meg ezeket a programrészeket esetről esetre!

Explicit bejárás

Explicit mozgási parancsok számára külön kezelő van, az Xmove. Ez a hasonló programrészek számára fenntartott programterület első kezelője, kódja az 5-1. ábrán látható.

```
NÉV:                  XMOVE
TIPUS:                KEZELO
FELADATA:             Explicit modon megadott
                      mozgas

200 D=CT(8)+CT(9)*10-1:FORK=1TO10:CT(5)=B(
K):GOSUB1000:IFD<>CT(8)ORCT(0)<>CT(6)+CT(7)
*10THENNEXTK:GOTO202:ELSEIFB(K)<>0THEN202EL
SEB=CT(9):GOTO206
202 D=D+1:GOSUB1120:IFA=22THENB=4:GOTO206:E
LSEIFA=23THENB=5:GOTO204:ELSEIFA=0THENB=6:G
OTO206:ELSECT(0)=A:CT(1)=CT(1)+1:GOTO100
204 GOSUB1100:GOTO580
206 GOSUB1100:GOTO104
```

5-1. abra. Az Xmove kezelo

Az előző fejezetből emlékezhetünk rá, hogy a parancsértelmező, miután beolvasott valamit, leválasztja az első szót és kikeresi a szótáblázatából. Mikor megtalálta, kikeresi a szóhoz tartozó AZ számot is. A számban benne foglaltatik a kezelő száma, amelyet ennek a parancsnak meg kell hívnia.

A szótáblázatban tizenhat olyan szó van, amelyek száma Xmove figyelmét igényli. Ezek a következők:

KÓD A	KÓD B	IRÁNY	KÓD A	KÓD B	IRÁNY
0	1	ÉSZAK	5	6	DÉLNYUGAT
1	2	ÉSZAKKELET	6	7	NYUGAT
2	3	KELET	7	8	ÉSZAKNYUGAT
3	4	DÉLKELET	8	9	FEL
4	5	DÉL	9	10	LE

5-2. ábra. Az irányok kódolása
Az A kódolást akkor használjuk, ha egy számjegyen kell az irányt kódolni

- Az iránytű nyolc iránya, rövidített formában: É, D, K, NY, ÉK, DK, ÉNY és DNY.
- A négy fő égtáj: ÉSZAK, DÉL, KELET és NYUGAT.
- A függőleges irányok rövidítéseikkel együtt: FEL, LE, F és L.

Ha ezt a tizenhat szót önmagában írjuk le, Xmove működésbe lép, mivel mindegyik AZ száma 01-gyel végződik, az Xmove kezelő számával.

(Mi történik, ha ezeket a szavakat ilyesféle szavakkal használjuk, mint „MENJ"? Egy „MENJ ÉSZAKNAK" jellegű parancs explicitnek számít az „ÉSZAKNAK" miatt. A „MENJ" szó azonban átmenetileg az implicit bejárási parancsok kezelőjéhez kerül. Ezt később látni fogjuk. Az explicit információt az irányt jelölő „ÉSZAK" szó hordozza.)

A szó AZ száma többet is tartalmaz, mint pusztán a kezelő számát. A 3. és 4. számjegyet arra tartottuk fönn, hogy járulékos információt tartalmazzanak, azért, hogy egy általános kezelő különböző szavakra más-más eredménnyel reagáljon. Az irányt jelölő szavaknál mindegyik AZ szám használja a 3. és 4. számjegyet, hogy átadja a kezelőnek, melyik irányról van szó. Együttesen ezekhez a számjegyekhez 1 és 10 közötti érték tartozik, az 5-2. ábrával összhangban.

Mint más szinoním jelentésű szavak esetén is, ha egy irányt jelző szó valaminek a rövidítése, akkor a kettőhöz azonos AZ szám tartozik. „DÉL" és „D" szinonimák, és mindkettőhöz a 10501 AZ szám tartozik. Az első 1-es azt jelenti, hogy mindkettő megengedhető első szóként. 05 a déli irányt jelenti, és a záró 01 behívja az Xmove kezelőt.

Ha az Olvasó visszalapoz az előző fejezet parancsértelmező kódjához, megfigyelheti, hogy amikor Xmove (vagy akármelyik másik kezelő) meghívására kerül sor, néhány információ már felhasználásra készen áll. Először is az N változó még mindig tartalmazza a beírt parancs 1. szavához tartozó AZ számot. Másodszor a CT(6) és CT(10) közötti változókban még mindig meg-

található N öt számjegye, szétválasztva. Harmadszor a TX$(3) és TX$(2) láncok változatlan formában tartalmazzák az 1. és 2. szót. Mindez elősegíti azt, hogy egy adott kezelő ellássa feladatát.

Xmove döntései

Az 5-1. ábrán az Xmove kezelő látható, ami valószínűleg a *Kardhalak és kincsek* legtöbbször foglalkoztatott kezelője. A kód magyarázatához felsoroljuk ellátandó feladatait.

- Ellenőriznie kell az akadálylistát, hogy nem korlátozza-e valami a mozgást a megadott irányban.
- Ellenőriznie kell a bejárási táblázatot, hogy a beadott irányú elmozdulás nem végzetes-e vagy lehetetlen.
- Végre kell hajtania az elmozdulást, ha lehet, és meg kell növelnie azt a számlálót, ami a megtett lépéseket tartja számon.

Az első feladat igen nehéz. Ha a játékos úgy dönt, hogy észak felé megy, esetleg akadály állja útját. Még a 2. fejezetben láthatta az Olvasó, hogy kétféle akadály van: aktív (pl. egy lény) és passzív (mint egy bezárt kapu). Ezeket a BK(*n*) vektor tartja nyilván, ami speciális számok segítségével megmondja, hogy hol vannak az akadályok, melyik irányt blokkolják, és még sok egyebet.

Az 5-3. ábra mutatja, hogyan vannak BK(*n*)-ben ezek a számok kiosztva. A szám 1. és 2. számjegye tartalmazza, hogy melyik helyiségben van az akadály. A 3. számjegy mondja meg, hogy az akadály milyen irányt blokkol (0-tól 9-ig terjedő számmal jelöljük a tíz lehetséges irányt). A 4. számjegy annak az üzenetnek a sorát tartalmazza, amit ki kell írni, ha belebotlunk az akadályba (az 1-es, 2-es, 3-as üzenet van fenntartva az akadályok számára).

Az 5. számjegy azt jelzi, van-e BK(*n*)-ben egy másik szám, amelyik ehhez kapcsolódik, és hol van (mint pl. egy kapu esetén, amelyik egyidejűleg két helyiségben van). Az ilyen jellegű párral rendelkező akadályszámok BK(*n*)-ben közvetlenül szomszédosak egymással; az 5. számjegy azt mondja meg, hogy a pár másik tagja ezt megelőzi, követi vagy nincs is. (Lény típusú akadályok csak egy helyiséget foglalnak el, ezért csak egy számot igényelnek a BK(*n*) vektorban.) Végül a szám előjele azt jelzi, hogy járható-e az akadály vagy sem. Ha a szám pozitív, az akadály nem járható; ha negatív, akkor járható. (A kapu pl. lehet nyitva vagy zárva.)

Tehát Xmove ismeri, melyik iránnyal próbálkozunk. Végig kell néznie BK(*n*) összes elemét! Ha nem talál olyan elemet, amelyben a helyiség száma egyezik, az elmozdulás lehetséges. Ha nem talál olyan elemet, amelyben az irány megegyezik a szándékolt elmozdulás irányával, az elmozdulás lehetséges. Ha azt találja, hogy az akadály járható, az elmozdulás lehetséges, de ha a három eset valamelyikében egyezést talál, nem enged tovább.

Xmove azzal kezdi, hogy ellenőrzi a helyiség és az irány egyezését az

ELŐJEL	5	4	3	2	1
AZ AKADÁLY ÁLLAPOTA + VAGY -	A HOZZÁTARTOZÓ BEJEGYZÉS HELYE 0 - 2	AZ AKADÁLY TIPUSA 1-3	A LEZÁRT IRÁNY 0-9	A HELYISÉG SZÁMA 1-20	

5-3. ábra. Az egyes számjegyek jelentése sz akadályok litájának BK(n) vektorában

akadálylistán, BL(*n*)-en. Emlékezzünk arra, hogy az irány az egyik része a szavak AZ számába beágyazott járulékos információnak. Az AZ szám még mindig az N változóban van, ezért Xmove-nak le kell választania az irányra vonatkozó információt.

Az Xmove-ban szereplő első kifejezés éppen ezt teszi. A CT(6) és CT(10) közötti változók még mindig az AZ szám 1-től 5.-ig terjedő számjegyeit tartalmazzák, ezen belül CT(8) és CT(9) tartalmazza az irány 1-től 10-ig terjedő értékét. A CT(8)+CT(9)*10 kifejezés ugyan előállítja ezt az értéket, de nekünk 0 és 9 közötti formában kell, mivel az akadálylista ezt a formát használja. A kiszámított értéket eggyel csökkentjük, és az eredményt D-be tesszük.

Ezután Xmove lefuttat egy ciklust, hogy a BK(*n*) vektor összes elemét megvizsgálja. Mivel a számokban szereplő bizonyos számjegyekkel összehasonlításokat kell végeznünk, minden egyes elemet szét kell bontani az Analyz szubrutin segítségével. Tehát a ciklus fog egy számot BK(*n*)-ből, elemzésre beteszi CT(5)-be, és meghívja Analyz-t, majd elvégzi a szükséges összehasonlítást. Xmove olyan FOR-NEXT ciklust használ, ahol a ciklusváltozó K; BK(*n*)-nek tíz eleme van; ez lesz a ciklus végértéke.

Valahányszor a ciklus kiválaszt egy elemet BK(*n*)-ből, Xmove ellenőrzi az elemet. Azonos-e a (D-ben tárolt) választott irány a blokkolt iránnyal (az akadályhoz tartozó szám 3. számjegyében vagy CT(8)-ban megtalálható-e)? Megegyezik-e a jelenlegi helyiség (ami mindig CT(0)-ban található) azzal, amelyikben az akadály van? (A CT(6)+CT(7)*10 kifejezés a helyiség számát állítja elő az 1. és a 2. számjegyből.) Ha nem, akkor a vizsgálóciklus továbblép az akadálylista következő elemére. Ha nem talál egyezést, a vezérlés a 202-es sorra adódik, amely az elmozdulással kapcsolatos egyéb korlátozásokat ellenőrzi.

Mi történik, ha megegyeznek az akadályok? Ekkor hátra van még egy végső kérdés: járható-e az akadály? Az biztos, hogy van itt egy kapu — de esetleg nyitva van! Ezt úgy ellenőrizzük, hogy megnézzük, kisebb-e a szám nullánál. Ha igen, az akadály járható, és figyelmen kívül lehet hagyni. Ha nem, az akadály gátol, és az elmozdulást lehetetlenné teszi.

Az akadály számának 4. számjegye tartalmazza annak az üzenetnek a számát, amit a nehézség magyarázataként ki kell írni. A *Kardhalak és kincsek* esetén ez a számjegy lényeknél 1, acélrácsoknál 2 és a kapuknál 3. A 2-es üzenet pl. így hangzik: „A RÁCS BE VAN CSUKVA ÉS LE VAN LAKATOLVA". A 206-os sor meghívja Mesprt-t, hogy a sort kiírassa, aztán visszatér a Vezérlőbe.

(Természetesen egy akadály járhatóvá tehető, ha ismerjük a módját. Kapuk esetében a NYISD, lények esetén az ÖLD parancsot a megfelelő fejezetben megbeszéljük.)

A bejárási táblázat ellenőrzése

Mindez nagyon szép és jó. Esetleg nem állja utunkat bezárt kapu. Most Xmovenak utána kel néznie a legilletékesebbnek, a bejárási táblázatnak, a helyszín térképének. Ebből a kezelő megmondhatja, hogy melyik helyiség lesz a végállomása a kívánt irányú elmozdulásnak, vagy hogy az irány valami borzalmas végzet felé vezet-e.

Az 5-4. ábra tartalmazza a tényleges bejárási táblázatnak egy kis részét, ez az 1-es adatblokk. Teljességében a táblázatnak húsz sora van, minden helyiséghez egy. Minden sorban tíz szám van, és egy tizenegyedik, amelyet az implicit bejárás kezelője használ.

```
5000 DATA1,2,2,1,1,1,1,1,0,3,9
5002 DATA2,2,2,2,2,1,1,2,0,8,9
5004 DATA0,0,4,10,0,0,0,0,1,0,3
5006 DATA0,5,0,0,11,0,3,0,0,0,4
5008 DATA0,0,0,0,0,4,0,0,0,0,5
5010 DATA0,0,0,12,0,0,0,0,0,23,3
5012 DATA0,0,0,14,0,0,0,0,0,0,3
5014 DATA0,0,0,0,14,0,0,0,2,0,8
5016 DATA9,0,16,15,9,0,0,9,0,0,7
.
.
.
```

5-4. ábra. A bajárási táblázat első néhány sora, ahogy a DATA blokkban megtalálható

Ez a tíz szám megfelel a tíz lehetséges iránynak. A számok annak a helyiségnek a számát jelölik, amelybe a megadott irányú elmozdulás eredményeként jutunk. Tehát amikor egy programrész tudni akarja, hogy hol köt ki a játékos, ha pl. a 3-as helyiségből délkelet felé megy, egyszerű dolga van. Rátalál a harmadik sorra (a 3-as helyiség miatt) és a negyedik számra (a délkeleti a negyedik irány). A táblázat szerint a játékos a 10-es helyiségbe jut, feltéve, hogy semmilyen akadály sem állja útját.

Várjunk csak egy percet! Mit jelentenek azok a nullák ott a sorban? Való igaz. Azontúl, hogy egy elmozdulás eredményeként egy másik helyiségbe jutunk, az is megeshet, hogy helyben toporgunk — mert a választott irányban fal van. A nulla helyiségszám tehát falat jelent, és ha ilyen irányban próbálkozunk az „ARRA NEM MEHET" üzenetet kapjuk. A lépés eredménye akár halál is lehet: a játékos lezuhan egy szikláról vagy egy lángfalba lép. A fel nem használt 22-es helyiségszám jelenti a lezuhanás általi halált, a 23-as a tűzhalált. Ha a játékos a bejárási táblázatban 22-vel vagy 23-mal jelölt irányba lép, meghal, és a Resur (resurrection = újjáélesztés) nevű különleges kezelő meghívásával nagy pontveszteség árán újjáéleszti a játékost.

A Travec (travel vector = bejárási vektor) nevű szubrutin megkönnyíti Xmove-nak az alapvető fontosságú számok kikeresését a bejárási táblázatból. Ha a D változóban egy 1 és 10 közé eső irányjelző szám van, akkor Travec kikeresi a táblázatnak a mostani helyiséghez tartozó sorából a célt jelző számot és az A változóba teszi. Xmove-nak már rendelkezésére áll az irányjelző szám D-ben 0 és 9 közé eső formában; egyet hozzáad D-hez és meghívja Travec-et. (Az Olvasó a 3. fejezetből ellenőrizheti Travec részletes működését, valamint Access közreműködését a szám megkeresésében.)

Miután A-t beállítottuk, a végcél szerint már csak egyszerűen össze kell hasonlítani a három különleges esethez tartozó számmal, 22-vel, 23-mal és 0-val. Az első két különleges esetben a halál bekövetkeztéről tudósító üzenetet kell kiírni: a B változó vagy a 4-es (tűzhalál), vagy az 5-ös (lezuhanás) üzenetre áll, és hívja Mesprt-t. Ezután a különleges halálesetet kezelő Resur meghívására kerül sor a 204-es sorban. A harmadik esetben (nulla) az „ARRA NEM MEHET" üzenet száma kerül B-be, és ismét Mesprt meghívása következik. A kezelő visszatér a Vezérlőparancs alárendelt részéhez, hogy új parancsot fogadjon.

Ha Xmove sikeresen kitért az összes különleges eset elől, akkor végre létrejön az elmozdulás. CT(0), a helyiség száma, megváltozik A-ra, a célnak megfelelően. Ugyanakkor a CT(1) változó eggyel nő. A CT(1) ugyanis az a számláló, amely a megtett lépéseket jegyzi föl, ez azoknak a játékosoknak érdekes, akik a legkedvesebb lépéssel akarnak átjutni a barlangrendszeren. Ezután Xmove befejeződik; visszatér a Vezérlőleíró alárendelt részéhez, hogy a kalandozó szétnézhessen az új helyszínen.

Belátjuk már, milyen bonyolult lehet az elmozdulás? És ez még csak az explicit bejárás!

Implicit bejárás

Az implicit bejárás azoknál a mozgást jelölő szavaknál jut szerephez, ahol nincs pontosítva a játékos szándéka. Ezekhez a szavakhoz az Imove (Implicit move = implicit elmozdulás) kezelő tartozik. Az 5-5. ábrán látható a teljes BASIC szubrutin.

```
NÉV:            IMOVE
TIPUS:          KEZELO
FELADAT:        Implicit modon megadott
                parancs

220 IFTX$(3)=""THEND=11:GOSUB1120:N=A*100+1
0101:GOTO108:ELSEA$=TX$(3):GOTO106
```

5-5. ábra. Az Imove kezelő

Az Imove a 2-es kezelő. A szavak táblázatában jelenleg négy olyan szó van, amelyek AZ száma Imove végrehajtását igényli. Ezek: BE, KI, MENJ, LÉPJ.

Elég különös, hogy mind a négynek ugyanaz az AZ száma. Jogosnak tűnhet a kérdés: honnan tudhatja Imove, hogy melyik irányra következtessen ezekből a szavakból, hiszen egyik sem ad további információt AZ számában.

A válasz a bejárási táblázatból olvasható ki. Emlékezzünk vissza, hogy minden helyiséghez tartozik egy sor, és minden sorban tíz szabályos irányt jelző szám van, no és egy fel nem használt tizenegyedik. Ez a tizenegyedik szám most jut szerephez, mint *feltételezhető irány*. Ez nem egy helyiség száma, hanem irányt jelölő szám 0 és 9 között. Minden esetben, amikor az irány nem egyértelmű, ezt a feltételezhető irányt használjuk.

Nézzük pl. az 1-es helyiséget, a mélyedést a föld felszínén, alján ott a föld alá vezető lyuk. Nyilvánvaló, hogy a feltételezhető irány lefelé van; így BE logikus elmozdulást eredményez; belépünk a lyukba. Való igaz, hogy sem a KI, sem a MENJ és LÉPJ nem passzol. Mondhatjuk, hogy a feltételezhető irány erősen korlátozza az implicit bejárást kezelő kód nagyságát. Enélkül azonban minden egyes helyiségben szükségünk lenne egy-egy külön számra minden egyes felhasznált implicit szóhoz. Szóba jöhetne esetleg néhány kompromisszszum, de többnyire az irány megválasztásának ez a módja elég jól beválik.

Érthetően Imove elég egyszerű! Három esetben kerül sor a végrehajtásra. Az első esetben az implicit bejárást jelentő szó egyedül is leírható, pl. MENJ.

A másodikban az implicit bejárást jelentő szó egy másik implicit bejárást jelentő szóval együtt írható be, pl. MENJ BE. A harmadik esetben az implicit bejárást jelentő szóval, pl. MENJ ÉSZAKNAK. Az Imove mindhárom esetet tudja kezelni.

Először az első esetet elemezzük. Ha a játékos csak azt írja: „MENJ", akkor van az 1. szó, és nincs 2. Azaz a TX$(3) lánc, amelyik a 2. szót tartalmazza, üres. Az Imove meggyőződik róla, hogy TX$(3) üres-e. Ha üres, akkor az Imove előkeresi a feltételezhető irány számát. Beállítja a D változót 11-re, meghívja Travec-et, és A-ba kerül a feltételezhető irány száma. Ebből a számból egy mesterséges AZ számot állít elő, mégpedig olyat, amilyen egy explicit bejárási szóhoz tartozik. Az A* 100+10101 kifejezés olyan AZ szót eredményez, ami már az Xmove explicit kezelőt igényli, és 1 és 10 közé eső irányt szab meg.

Legvégül az Imove bejuttatja ezt a mesterséges AZ számot a Vezérlőbe a 108-as sornál. Ezen a ponton a Vezérlő úgy cselekszik, mintha explicit bejárási parancsot kapott volna, és ennek megfelelően halad tovább.

Ha két szót írtak be, a második szó A$-ba kerül. A 106-os sornál lép be újra a Vezérlőbe. Itt a Vezérlő úgy viselkedik, mintha csak egy szó, a második lett volna beírva. A „MENJ ÉSZAKNAK" parancs esetén a Vezérlő most azt látja: „ÉSZAKNAK", és nem okoz neki gondot, hogy mit tegyen. A „MENJ BE" parancsból a Végrehajtó azt látja, hogy „BE", és végül ismét meghívja Imove-t, amelyik a feltételezhető irányt használja.

Az eddigiek elintéznek a kalandprogram lehetséges bejárási módja közül kettőt. Hátra van még egy.

Mágikus bejárás

A mágikus bejárásnak az a szerepe a kalandprogramban, hogy segítsen a játékosnak egy csapdából kijutni, vagy módot adjon rá, hogy a lehető legkevesebb lépéssel fejezze be a játékot. Eredetileg a mágikus utazás azt a lehetőséget biztosítja a kalandozónak, hogy kijátssza a helyszínen történő mozgás szokásos szabályait, és egy hatalmas ugrással egy távoli helyiségbe jutton úgy, hogy figyelmen kívül hagyja az útjában álló esetleges falakat vagy akadályokat.

Ezt a fajta bejárást valamilyen mágikus szóval érjük el. A kalandprogramban rejlő kihíváshoz tartozik az is, hogy kitaláljuk, létezik-e mágikus utazás, és milyen szó váltja ki. A szó talán valamelyik helyiség falára van felírva, esetleg egy könyvben van. Netán valamelyik lény mondogatja időnként. Bármelyik eset áll is fenn, a szó valahol el van rejtve és elő kell ásni.

A *Kardhalak és kincsek*-ben, mint látni fogjuk, a bűvös szó egy rövid versbe van beleírva a 6-os helyiség falán. Ha a játékos leírja az OLVASD parancsot, megismerheti a verset. A bűvös szó AARDVARK (ne kérdezze az Olvasó, miért — egyszerűen jól hangzott). Kétféle módon lehet felhasználni. A játékos leírhatja a „MONDD AARDVARK" parancsot, és a mágikus utazás elkezdődik, vagy egyszerűen leírja, hogy „AARDVARK", mert így is működik. Ahogy

hamarosan meglátjuk, a szó hatékonyságát bizonyos körülmények természetesen korlátozzák.

Rögtön a legelején láthatjuk, hogy három kezelő kell a leírt mágikus utazás megvalósításához. Szükség van:

- egy kezelőre, amelyik felismeri az OLVASD szót;
- egy kezelőre, amelyik felismeri a MONDD szót;
- egy kezelőre, amelyik felismeri az AARDVARK szót.

Elsőnek a READ nevű kezelőt vizsgáljuk meg. A 400-as sorban kezdődik. Listája a 4-5. ábrán látható.

Két esetbe lehet használni az OLVASD parancsot: a 6-os helyiségben vagy másutt (egyszerű ugye?!). A 6-os helyiség leírásában a játékosnak tudomására jut, hogy egy jós „ÜZENETET HAGYOTT A FALON". Egyébként sehol sincs olvasnivaló. Az OLVASD parancsra csak kétféle választ várhatunk. A 6-os helyiségben van, a játékos megtudja a verset, máshol viszont nem hall semmi érdekeset. Ebből következik, hogy a READ kezelőnek fel kell ismernie, hol is van a kalandozó, és fel kell készülnie, hogy a helytől függően két üzenet egyikét írja ki.

A READ 402-es sora meghívja a Mesprt szubrutint, hogy a megfelelő üzenetet kiírja. A 400-as sor kiválasztja az üzenetet. Ha CT(0), vagyis az aktuális helyiség száma 6, akkor a 33-as üzenet, maga a vers jelenik meg. Bármely más helyiségből a 32-es üzenetet kapjuk: „NINCS ITT SEMMI OLVASNIVALÓ... ÓH DE UNALMAS!"

```
NÉV:          READ
TIPUS:        KEZELŐ
FELADATA:     Különleges üzenetek
              kiolvasása

400 IFCT(0)<>6THENB=32:GOTO402:ELSEB=33
402 GOSUB1100:GOTO104
```

5-6. ábra. A Read kezelő

A kiírt vers ugyan nem egy remekmű, de a célnak megfelel:

A VESZÉLY
ITT NEM CSEKÉLY
DE MONDD AARDVARK
EZ A SEGÉLY

Mellesleg, a vers a többi üzenethez hasonlóan DATA utasításként van tárolva. Hogyan lehetséges akkor, hogy négy takaros verssor formájában jelenik

```
NÉV:            SAY
TIPUS           KEZELŐ
FELADÁT:        Bűvös szavak kimondása

460 IFLEFT$(TX$(3),5)<>"AARDV"THENB=34:GOSU
B1100:GOTO104:ELSE560
```

5-7. ábra. A Say kezelő

meg? A titok nyitja a DATA sor beírásában rejlik. A TRS–80 „le" nyíl karaktere egy újsor-karaktert szúr a szövegbe. Amikor a 33-as üzenethez érünk, a programozó írjon egy új sort mind a négy verssorba. A DATA utasítást ez nem zavarja, de a végeredmény kiírásakor mutatós.

Most vessünk egy pillantást a SAY kezelőre! Először is, ha a játékos azt írja, „MONDD AARDVARK", a kezelőnek úgy kell reagálnia, mintha a játékos a bűvös szót önmagában írta volna le, és kezdeményeznie kell e mágikus utazást. Másodszor, ha a játékos azt írja, „MONDD XYZ", és XYZ bármi, de nem a bűvös szó, semmi se történjen, és jelenjen meg egy üzenet.

A második eset ellenőrzése megtörténik az 5-7. ábrán a SAY kezelő elején. Amikor a játékos azt írja, „MONDD AARDVARK", a második szót, az „AARDVARK"-ot őrzi meg TX$(3). A kezelő megnézi a második szó első öt betűjét, hogy lássa, ráillenek-e erre az esetre. A LEFT$(X$,*n*) BASIC kifejezés használatának az a célja, hogy leválassza TX$(3)-ból a kívánt betűket.

Abban az esetben, ha nem egyeznek (és jaj annak a játékosnak, aki elírja „AARDVARK"-ot), meghívjuk Mesprt-t, hogy kiírja a 34-es üzenetet, amelyik így hangzik: „SEMMI SEM TÖRTÉNIK". Aztán ismét belépünk a Vezérlőbe.

Figyeljük meg, hogy ez az üzenet szándékosan semmitmondó! Nem mondja, hogy „NEM ISMEREM EZT A SZÓT", akkor sem, ha második szó hiányzik a szótáblázatból. Célja, hogy kétségek között maradjon a játékos: vajon hasznos lehet-e még a második szó. A tapasztalt játékosok ugyanis minden új kalandjátékban kipróbálják a hasonló játékokból összeszedett régi bűvös szavakat. Így be fogják írni, hogy „MONDD ABRAKADABRA" vagy „MONDD SZEZÁM TÁRULJ", vagy valami hasonlót. Mivel a kezelő csak azt állítja, hogy semmi sem történt, lehetséges (gondolja a játékos), hogy a parancs egy másik helyiségben vagy más körülmények között működik. Ez a fajta kétértelműség meghosszabbítja a játék rejtélyeit.

Mi történjen a „MONDD AARDVARK" hatására? Ekkor a kezelő továbbhalad, és az 560-as sorra ugrik, ahol újra egy másik kezelő kezdődik:

```
NÉV:                  AARDVARK
TIPUS:                KEZELÖ
FELADATA:             A buvös szo kezelése

560 IFCT(0)=6THENCT(0)=1ELSEIFCT(0)=1THENCT
(0)=6ELSEB=34:GOSUB1100
562 GOTO100
```

5-8. ábra. Az Aardvark kezelő

az ún. AARDVARK kezelő (l. az 5-8. ábrát). Az 560-as és az 562-es sor határozza meg végül, hogy részt vesz-e a játékos a mágikus utazásban vagy sem.

A mágikus utazás korlátozásai kalandprogramról kalandprogramra változnak. Egyes játékokban a játékosnak rendelkeznie kell egy bizonyos tárggyal ahhoz, hogy utazhasson. Másokban csak bizonyos helyiségből indulhat. Néhány játékban az utazás véletlenszerűen választott távolsági utazást jelent, másokban a bűvös utazás két előre meghatározott helyiség között oda–vissza útra korlátozódik.

A *Kardhalak és kincsek*-ben a mágikus utazás csak két helyiség között zajlik: a 6-os és az 1-es helyiség között. Ez két szempontból jelent segítséget. Először: a 6-os helyiségben egy veszélyes lény van, amelyik a helyiségből kivezető egyetlen kaput őrzi. Ha a játékos betéved a helyiségbe, kincset talál, de egyben csapdát is! A helyiségből kivezető egyetlen út a mágikus utazás. Másodszor: az 1-es helyiség a mélyedésben levő helyiség alja, egyben a *Kardhalak és kincsek* támaszpontja.

Minden talált kincs csak akkor számít be a játékos pontjaiba, ha a föld alatti barlangból kell fölcsempészni az 1-es helyiségbe. Kapóra jön tehát a támaszpontra vezető mágikus utazás; hisz a lassúbb módszer, a „gyalogos", rengeteg kockázattal jár, belebotolhatunk pl. egy kiéhezett fenevadba.

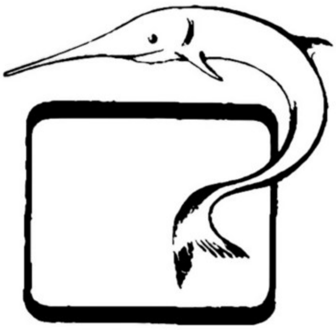
Az AARDVARK kezelő hozza létre e két helyiség között az utazást. A CT(0) aktuális helyiség számának ellenőrzésével az AARDVARK eldönti, milyen irányú legyen az utazás. Ha a kalandozó a 6-os helyiségben van, átkerül az 1-es helyiségbe úgy, hogy egyszerűen megváltoztatjuk CT(0) értékét.

Az 1-es helyiségből pedig át lesz szállítva a 6-osba. Mi történik, ha nem az 1-es vagy 6-os helyiségben van, hanem valahol másutt? Ebben az esetben a kétértelmű „SEMMI SEM TÖRTÉNIK" üzenet jelenik meg. A játékos ismét ott áll a rejtély előtt: vajon milyen körülmények között működik az „AARDVARK" szó?

Tekintsük át az utazást!

Összegezve: láthattuk a kalandozó rendelkezésére álló háromfajta utazást és a hozzájuk kapcsolódó kezelőket. Ezek: az Xmove kezelővel végrehajtott explicit utazás, az Imove kezelővel megvalósított implicit utazás, valamint a SAY és AARDVARK kezelőkkel megvalósított, valamint a READ kezelővel támogatott mágikus utazás.

Az élethű utazási feltételek alkotják a hihetőség egyik részét a kalandban. A másik részt a tárgyakkal való kölcsönhatás érezteti. A következő fejezetben látni fogjuk, hogyan valósul meg ez a kölcsönhatás a *Kardhalak és kincsek*-ben.



6. FEJEZET

Megváltoztatjuk a helyszínt

Milyen érzés lenne, ha egy idegen lakásban járkálnánk és megpróbálnánk valamit felvenni, de az nem mozdulna? Mikor már úgy érezzük, hogy jól megragadtuk a képes újságot és felemeltük... a helyén marad. Azután megpróbálunk elmenni, kezünket kinyújtjuk, hogy a bejárati ajtót kinyissuk... de nem nyílik. Micsoda lidérces álom?

Az a világ, amelyben semmi sem változtatható meg, nem valós világ. A kalandprogram úgy tartja fenn mesterséges világát, a valóság látszatát, hogy a benne mozgó kalandozó változásokat hozhat létre. A kapuk nyílnak és záródnak; a tárgyak elmozdíthatók.

Kétfajta parancs szükséges a realitás látszatához. Ezek a következők:

- a kapuk kinyitásához, becsukásához és bezárásához szükséges parancsok;
- a tárgyak felvételéhez, szállításához, valamint lerakásához szükséges parancsok.

Mindegyik parancshoz kezelők, táblázatok és tömbök tartoznak, amelyeket a parancsok megváltoztatnak, nevezetesen a tárgyak állapottömbje és az akadályok listája.

Zárt kapuk mögött

A kapukat nyitó és záró kezelők működésének megértéséhez röviden ismételjük át az akadályok listáját. Emlékezzünk rá, hogy a BK(n) vektor egy sorszámot tartalmaz, ez a kalandozó előrejutását gátoló akadályokat írja le.

Háromfajta akadály van; kapuk, acélrácsok és lények. A kapuk és az acélrácsok kezelőit most beszéljük meg; a lényekkel csak harcban bánhatunk el, ahogy ezt a következő fejezetben leírjuk.

A kapuk és rácsok egyediek annyiban, hogy egyszerre két helyiséget foglalnak el. Ezért minden kapuhoz vagy rácshoz a BK(n) vektor két elem tartozik,

egy-egy az akadály állapotáról mindkét helyiségben. Ha egyszer egy kapu be van csukva és le van lakatolva, ez akadályt jelent a kalandozónak, függetlenül attól, hogy a kapu melyik oldalán van. Azaz bármilyen kezelő nyitja és zárja a kapukat és a hasonlókat, képesnek kell lennie rá, hogy BK(n)-ben megváltoztassa a kapuhoz tartozó mindkét állapotjelző számot.

A *Kardhalak és kincsek*-ben, csakúgy, mint a hasonló kalandprogramokban, van egy kulcs (a 11-es tárgy), amelyik a kapukat és rácsokat nyitja. E nélkül a kulcs nélkül az akadályok állapota BK(n)-ben nem változtatható meg. Azonban más programoktól eltérően a kapuknak és rácsoknak csak két állapota van: zárt és lelakatolt vagy nyitott. Más programok lehetővé tesznek olyan átmeneti állapotot, mint „becsukott, be nem lakatolt", de ez a látszólag egyszerű bővítés elég nagy mértékben bonyolítja az akadályok kezelését. (Ez persze ne tartsa vissza az Olvasót a megvalósításától, ha úgy érzi, hogy megéri a vesződséget!)

Először tekintsünk egy képzeletbeli kezelőt, amelyik kapukat nyit ki!

Egy ilyen kezelőnek a következő kérdésekre kell válaszolnia:

- Megmondta a játékos, mit akar kinyitni?
- Ha megmondta, az egy közeli kapu vagy rács?
- Ha igen, zárva van a kapu, vagy a rács?
- Ha zárva van, a játékosnál van a kulcs?

Az 5-ös számú Open nevű kezelő válaszol a fenti kérdésekre és nyitja a kaput. A 6-1. ábrán látható az Open listája. A szavak táblázatában két szó igényli az Open végrehajtását. Név szerint TÁRD és NYISD. Ennek így van értelme, mert ha ebben a programban kinyitjuk egy kapu zárját, akkor a kapu ki is tárul, és ha azárat bezárjuk, abból következik, hogy be is van csukva. Ezért a két szót szinonimaként kezelhetjük.

```
NÉV:            OPEN
TIPUS:          KEZELO
FELADAT:        Kapuk és rácsok nyitása

280 IFTX$(3)=""THENB=7:GOTO284:ELSEA$-TX$(3
):GOSUB1080:CT(5)=N:GOSUB1000:A=CT(8):GOSUB
1200:IFA=0THENB=12:GOTO284:ELSEIFBK(A)<0THE
NB=13:GOTO284:ELSEIFOB(11,1)<>21THENB=16:GO
TO284:ELSEGOSUB1220:B=12+CT(9)
284 GOSUB1100:GOTO104
```

6-1. ábra. Az Open kezelő

Open azzal kezdi, hogy ellenőrzi, vajon elég információt adott-e a játékos ahhoz, hogy értelmesen járjon el. Ha a játékos csak annyit írt: „NYISD", az nem biztos, hogy elég; esetleg két kapu is nyílik egy helyiségből. Open ezt az esetet úgy ellenőrzi, hogy megnézi a TX$(3)-ban található 2. szót. Ha TX$(3) hossza nulla, akkor Open nem vesztegeti arra az időt, hogy továbbhaladjon. Inkább kiadja a 7-es számú „tettessünk süketet" üzenetet úgy, hogy a B változót 7-re állítja és a 284-es sorban meghívja Mesprt-t. A 7-es üzenet egyszerűen megkérdi, hogy „MIT MOND?", és lehetőséget ad a játékosnak, hogy világosabban fejezze ki magát.

Ha feltételezzük, hogy a játékos a „TÁRD" vagy „NYISD" kulcsszóval együtt beírt egy második szót is, akkor a kezelő megpróbálja kideríteni a 2. szó értelmét. Meghívja az 1080-as soron kezdődő Idword szubrutint. Idword fogja az A$ szöveges változóban tárolt szót, és átnézi a szavak táblázatát. Ha megtalálja, visszatér az *N* változóba a szó AZ számával. Ha nincs a program szótárában, úgy tér vissza, hogy *N* értéke nulla. Open átmásolja a 2. szót A$-ba és ráereszti Idwort-öt.

Mikor Idword befejeződik, Open-t az *N*-ben tárolt AZ szám egyes számjegyei érdeklik. Tehát meghívja Analyz-t, hogy bontsa számjegyeire *N*-t. Az Analyz fogja CT(5) tartalmát, és az 1.-től 5.-ig számjegyeket beteszi a CT(6)-tól CT(10)-ig terjedő változókba. Az Open CT(5)-be teszi *N*-t, a többit elvégzi Analyz. Jegyezzük meg, pillanatnyilag, hogy ha *N* nullával egyenlő (mert a második szó nem volt benne a szavak táblázatában), akkor Analyz egyszerűen nullát tesz a CT(6) és CT(10) közötti összes változóba.

Most, hogy Open-nek rendelkezésére áll az összes lecsupaszított számjegy, csak egy érdekli: a 3. számjegy. Emlékezzünk rá, hogy tárgyaknál a tárgy AZ számának 1. és 2. számjegye a tárgy sorszáma. Pl. ha a „PÓK" szót keressük ki a szavak táblázatából, az AZ szám 1. és 2. számjegyének értéke 15, mivel a pók a 15-ös számú tárgy a *Kardhalak és kincsek* tárgyainak listáján.

A kapu és rács jellegű tárgyak azonban speciálisak. Nem tárgyak a szó hagyományos értelmében; nem lehet őket elvini vagy elejteni. Ezért egy kapuhoz vagy rácshoz nem tartozik tárgysorszám. Helyette a 17-es különleges tárgyszám tartozik hozzájuk. A „KAPU" és „RÁCS" szó AZ számában az 1. és 2. számjegy értéke 17. Később látni fogjuk, hogy ha a játékos megpróbál felemelni és elszállítani valamit, aminek a tárgyszáma 17, azt a program visszautasítja, mondván: „NEM LEHET ELMOZDÍTANI". Ez elejét veszi néhány nagyon zavaró ellentmondásnak!

Ha a 2. szó AZ számában, amit Open épp most elemzett, nincs használható tárgysorszám, akkor mire jó egyáltalán az AZ szám? A többi számjegynek nincs is semmi jelentése, vagy mégis? A válasz, hogy igenis van. Emlékezzünk rá, hogy háromféle tárgy van! Az Open számára nagyon hasznos lehet, ha az AZ szám tartalmazza az akadály fajtáját. Csak a „KAPU" és „RÁCS" szó esetében az AZ szám 3. számjegye az akadály típusát jelenti: 2, ha rácsról és 3, ha kapuról van szó. (Az 1-es típus lény, de lényeket nem nyitogatunk, csukogatunk.)

Az Open-t csak azért érdekli az akadály típusa, mert az akadály típusát jelző számot használják a BK(n) akadálylista elemei. Az Open-nek két kérdésre kell válaszolnia az akadálylista segítségével: (1) van egyáltalán akadály ebben a helyiségben és (2) ha van, azonos-e azzal az akadállyal, amit a játékos ki akar nyitni vagy lakatolni?

Az akadályok listájának minden elemében benne van a válasz mindkét kérdésre. Minden elem 1. és 2. számjegye annak a helyiségnek a száma, ahol az akadály van, és a 4. számjegy a típus száma, 1, 2 vagy 3. Most tehát az Open kezelőnek át kell néznie az akadálylistát és két összehasonlítást kell elvégeznie. Először meg kell találnia azokat az elemeket, amelyek egyeznek a jelenlegi helyiség számával, ami (mint mindig) CT(0)-ban van. Másodszor, ezekből az elemekből ki kell keresnie azokat, amelyeknél az akadály típusa megegyezik a jelenleg CT(8)-ban található típusszámmal, azzal az akadállyal, amit a játékos írt be a NYISD parancs második szavaként.

Van egy ügyes szubrutin az akadálylista átnézésére. Neve Ckobs (check obstacles = ellenőrizd az akadályokat) és a 6-2. ábrán látható. Lényegében veszi BK(n) minden egyes elemét, felbontja számjegyeire és elvégzi ezt a két összehasonlítást. Ha talál ilyen elemet, úgy tér vissza, hogy az elem helyzetét az A változóba teszi. Ha nem talál egyező elemet, A értéke nulla lesz. A értékének felhasználásával Open megtalálja és meg tudja változtatni a BK(n) akadálylista megfelelő elemeit.

Ckobs egy 1-től 10-ig futó FOR-NEXT ciklussal kezdődik, mivel BK(n)-nek tíz eleme van. Minden elemet részeire szedünk Analyz meghívásával (GO-SUB 1000 segítségével). Előzőleg az Open kezelő beállította az A változó értékét a keresett akadály típusára. Tehát Ckobs összehasonlítja a helyiség

```
NÉV:                CKOBS
TIPUS:              SZUBRUTIN
BEMENŐ ADAT:        A=az akadály tipusa(1-3)
EREDMÉNY:           A=az elem sorszama az
                    akadályok listáján, ha van
                    A=0 ha nincs ilyen

1200 FORQ=1TO10:CT(5)=BK(Q):GOSUB1000:IFCT(
6)+CT(7)*10<>CT(0)ORCT(9)<>ATHENNEXTQ:A=0:E
LSEA=Q
1202 RETURN
```

6-2. ábra. A Ckobs szubrutin

számát és az akadály típusát BK(*n*) összes elemével. A CT(6) + CT(7)*10 kifejezés az akadálylista elemének 1. és 2. számjegyéből újra előállítja egy helyiség számát.

Ha ez az érték nem egyezik a CT(0)-ban található jelenlegi helyiségszámmal, vagy a CT(9)-ben található 4. számjegy eltér az A változóban tárolt akadály típusától, a FOR-NEXT ciklus folytatja a keresést. Ha a ciklus lefut anélkül, hogy egyezést találna, A értéke 0-ra lesz beállítva, és a szubrutin visszatér. Ha egyezést talál, akkor A értéke egyenlő lesz O-val, a FOR-NEXT ciklus változójával. Tehát ha a negyedik elem egyezik, A egyenlő lesz 4-gyel.

Itt az ideje, hogy válaszoljunk egy kérdésre, amit az Olvasó joggal tart számon. Röviddel ezelőtt átnéztünk egy táblázatot, hogy a 2. szót megkeressük. Ha egy szót nem találunk meg a szavak táblázatában, azt a szót az Open nem is tudja ellenőrizni. Ha a játékos ilyesmit ír le: „NYISD KI AZ UBORKÁT", mi tartja vissza a kezelőt attól, hogy hibásan reagáljon?

Ezt a Ckobs zárja ki. Emlékezzünk arra, hogy ha egy szót nem talál a szavak táblázatában, akkor az Idword nullát ad eredményül. Ez felbomlik öt számjegyre. Mikor az Open meghívja Ckobs-t, hogy végezze el a helyiség és az akadály típusának összehasonlítását, az egyezés kizárt. Miért? Azért, mert egy fel nem ismert második szó nullás típusú akadályt kívánna. Ilyen akadály pedig nincs; BK(*n*) egyik elemében sem egyenlő az akadály típusa nullával. Tehát egy „NYISD KI A KENGURUT" jellegű parancsra ugyanaz a válasz, mint egy „NYISD KI A KAPUT" parancsra egy kapu nélküli helyiségben.

Az Open kezelőnek most rendelkezésére áll az akadálylistában egy 1 és 10 közé eső sorszám, vagy nulla, ha nincs olyan elem, amely megfelelne a parancs kívánalmainak. Open hozzálát a cselekvéshez az új információ birtokában. Mi legyen, ha nem létezik ilyen akadály? Ha így áll a helyzet, A értéke nulla. Open ellenőrzi ezt, és meghívja Mesprt-t, hogy írja ki a 12-es üzenetet, amelyik így szól: „NEM LÁTOK ITT SEMMI HASONLÓT!"

Ezután Open-nek el kell döntenie, ki kell-e nyitni a kaput vagy rácsot. Nyilvánvaló, hogy ha már szabadon „leng a szélben", nevetséges lenne, ha a kezelő újra végigcsinálná a nyitás teljes folyamatát. Open az akadályok listáján szereplő elem segítségével dönti el ezt a kérdést. Most A megegyezik a megtalált elem sorszámával, és BK(A) maga az elem. Az akadályok listáján az elem előjele jelzi, hogy egy akadály járható-e, vagy sem. Azaz, ha az elem negatív szám, akkor az akadály járható; a kapu vagy rács ki van nyitva. Open megnézi, hogy BK(A) kisebb-e, mint nulla, és ha igen, akkor meghívja a 13-as üzenet kiíratását, amely így szól: „NEM SZÜKSÉGES".

Az utolsó eshetőség a kulcs birtoklása. A kulcs nélkül, ez a 11-es tárgy, egyetlen kapu vagy rács sem nyitható ki. A kulcsnak a játékos birtokában kell lennie; azaz magával kell vinnie. Nem heverhet egyszerűen csak valahol a közelben a helyiségben. Open ellenőrzi a tárgyak állapotmátrixát, hogy megtalálja a kulcsot. Az OB(11,1) változó elárulja, melyik helyiségben van jelenleg a kulcs.

A játékos hátizsákjához a 21-es helyiségszám van hozzárendelve. Más szó-

```
NÉV:              REVOBS
TIPUS:            SZUBRUTIN
BEMENO ADAT:      A=az akadalyok listaján
                  beluli sorszám
EREDMÉNY:         a megadott elemnek és a
                  párjának az állapot el-
                  lenkezojére változik

1220 BK(A)=-BK(A):CT(S)=BK(A):GOSUB1000:IFC
T(10)=1RETURNELSEBK(A-1+CT(10))=-BK(A-1+CT(
10)):RETURN
```

6-3. ábra. A Revobs szubrutin

val a játékos csak abban az esetben nyithat ki egy kaput vagy rácsot, ha az OB(11,1) egyenlő 21-gyel. Ha nem az, akkor a kezelő meghívja a 16-os üzenet kiírását, ez így hangzik: „NINCS KULCSA!”

Végül Open-nek sikerül az összes lépést végrehajtania, és készen áll rá, hogy kinyissa a kaput vagy rácsot. Ehhez meghívja a 6-3. ábrán látható Revobs (reverse obstacle = akadály átfordítása) szubrutint. Az A változóban egy 1 és 10 közé eső sorszám van az akadálylistáról; a Revobs összesen két feladatot lát el: átfordítja annak az elemnek az előjelét, amire A mutat, és ha akad a listán párja, akkor az előjelet is átfordítja.

Megfigyelhetjük tehát, hogy Revobs az elem előjelét átfordítja. Azaz, ha a kapu vagy rács zárva volt, kinyílik. A Revobs képes rá, hogy nyitott kapukat bezárjon. A Revobs nagyon jól jön, ha „ZÁRD BE A RÁCSOT” jellegű parancsot írnak le. Figyeljük meg azt is, hogy megtalálja az elem párját (ha van) és azt is megfordítja. Ily módon egy kapu mindkét oldalon kinyílik. Ha az elemnek nincs párja (ez a helyzet a lény típusú akadályoknál), a Revobs csak az első feladatot hajtja végre. A Revobsot még felhasználjuk a nem járható akadályok járhatóvá tételére a következő fejezetben, a támadó lényekkel való harc tárgyalásakor.

Az első feladat egyszerű. Az A változóban már benne van az akadálylistán szereplő elem sorszáma. Ezért Revobs megváltoztatja BK(A) előjelét. A második feladat némi okoskodást igényel. Az akadálylistán szereplő elemek 5. számjegyének az a feladata, hogy megmondja, van-e az elemnek párja, és ha igen, akkor hol van. Emlékezzünk arra, hogy egy kapuhoz vagy rácshoz tartozó két elem az akadálylistán mindig szomszédos egymással! Az 5. számjegynek megfelelően három lehetőség van, ezeket 0-tól 2-ig terjedő számjegyek jelölik a következő módon:

0. Az elem párja közvetlenül ez előtt az elem előtt van.
1. Az elemnek nincs párja; egyedül álló elem.
2. Az elem párja közvetlenül követi ezt az elemet.

A 0, 1, 2 számjegyeket nem önkényesen választottuk. Revobs már tudja, hogy a BK(A) elemet kell megváltoztatni. Most vagy BK(A−1)-et, vagy BK(A+1)-et, vagy egyiket sem kell megváltoztatni. Revobs felhasználhatja a 0, 1, 2 számjegyeket az elemhez tartozó másik elem azonosítására, ha van ilyen.

Tanulmányozzuk a 6-3. ábrát, hogy lássuk, hogyan is történik ez! CT(10)-ben van az 5. számjegy: 0, 1 vagy 2. REVOBS visszatér, ha ez 1, mert már megfordította BK(A) előjelét. Egyébként Revobs megváltoztatja a BK(A−1+CT(10)) elem előjelét, ez a hozzá tartozó elem, akár előtte, akár mögötte van. Ezzel a szubrutin befejeződik és visszatér az őt meghívó Open-be.

Felvetheti valaki a kérdést, jobb, ha most tisztázzuk: csak tíz elem van az akadálylistán, miért nem használunk az 5. számjegyen egy 0 és 9 közötti számot a tíz elemnek megfelelően? Ebben az esetben egy elem pontosan megmondhatná, hol van a párja, és nem lenne szükség rá, hogy az elem párja pont szomszédos legyen a másikkal.

Igen ám, de erre az a válasz, hogy ez automatikusan tíz elemre korlátozza az akadálylista méretét, és nem hagy teret a bővítésnek. A *Kardhalak és kincsek* jelenlegi változatában ugyancsak két kapu, egy rács és négy lény jellegű akadály van, ami bizony elég kevés. A most alkalmazott módszer, hogy az összetartozó elemekhez szomszédos helyeket használunk, lehetővé teszi, hogy az akadálylista akkora legyen, amekkorára szükség van.

Miután Revobs lefutott, az Open kezelőnek még egy utolsó feladata van hátra, tudatni kell a kalandozóval, hogy a kapu vagy a rács kinyílt. Az akadálylistán szereplő elem 4. számjegye egy 1 és 3 közötti szám, és azt jelzi, hogy milyen jellegű akadály lett megváltoztatva. A tárban az üzenetek blokkjában a kapuk és a rácsok kinyitását tudató üzenetek egymás mellett találhatók, és hogy a dolgokat egyszerűbbé tegyük, épp a megfelelő sorrendben. A 2-es típusú akadály rács, a 3-as típusú pedig kapu; ezért a rács kinyitásához tartozó üzenet megelőzi a kapuét. A 12+CT(9) kifejezés eredménye 14 a rács esetén és 15 a kapu esetén. A 14-es üzenet közli: „A RÁCS CSIKOROGVA KIESIK HELYÉBŐL". A 15-ös üzenet pedig így szól: „A KAPU SZÉLESRE TÁRUL". Figyeljük meg, hogy az üzenet azonos, akár azt a parancsot írtuk le, hogy „TÁRD KI A KAPUT", vagy csak „NYISD KI A KAPUT". Mindkét parancs eredménye azonos.

Zárjuk be a kaput magunk mögött!

A kapukhoz és rácsokhoz kapcsolódó másik kezelő a Close. Ez a 6-os kezelő és a 6-4. ábrán látható. A szavak listáján „CSUKD" és „ZÁRD" AZ száma indítja el Close végrehajtását.

Néhány egyszerűsítéstől eltekintve, Close sok tekintetben úgy működik, mint Open. Close-nak a következő kérdésekre kell válaszolnia:

- Megmondta a játékos, mit akar bezárni?
- Ha igen, közel van az a kapu vagy rács?
- Ha igen, nyitva van a kapu vagy a rács?

A sasszeműek biztosan észrevették az Open és a Close közötti egyetlen lényeges különbséget (a végeredménytől eltekintve). Ez pedig a kulcs szükségessége. Ahhoz, hogy egy kaput vagy rácsot becsukjon és bezárjon, a kalandozónak nincs szüksége kulcsra. Egyszerűen bezárul, és amint a kísérő üzenet mondja, „A ZÁR BEKATTAN".

```
NÉV:              CLOSE
TIPUS:            KEZELO
FELADAT:          Kapuk és rácsok zárása

300 IFTX$(3)=""THENB=7:GOTO304:ELSEA$=TX$(3
):GOSUB1080:CT(5)=N:GOSUB1000:A=CT(8):GOSUB
1200:IFA=0THENB=12:GOTO304:ELSEIFBK(A)>0THE
NB=13:GOTO304:ELSEGOSUB1220:B=17
304 GOSUB1100:GOTO104
```

6-4. ábra. A Close kezelő

Első döntését Close a 2. szó megvizsgálása után hozza. Ha nem létezik a 2. szó TX$(3)-ban, a „MIT MOND?" üzenet jelenik meg. Egyébként Close veszi a TX$(3)-ban található szót, és az A$ szöveges változóban átadja Idwordnek. Idword úgy tér vissza, hogy a szó AZ száma az N változóban van. Close meghívja Analyz-t, hogy szétválassza az AZ szám öt számjegyét. Ezután veszi a 3. számjegyet, az akadály típusát, és Ckobs-szal megvizsgáltatja, hogy a 2. szó által jelzett akadály valóban a helyiségben van-e vagy sem. Ha nincs (amint ezt az A változó nulla értéke jelzi), akkor a 12-es üzenet jelenik meg: „NEM LÁTOK ITT SEMMI HASONLÓT!" Végül az elem előjelét ellenőrzi. Ha pozitív, akkor a kapu vagy a rács már be van zárva és le van lakatolva, és a 13-as üzenet közli a játékossal: „NEM SZÜKSÉGES".

Ha a beírt parancs értelmesnek bizonyul mindhárom ellenőrzés után, akkor Close továbbhalad és a Revobs szubrutin segítségével megfordítja az akadály állapotát. A nyitott kapu vagy rács zárt állapotba kerül oly módon, hogy az akadálylista elemének és a hozzá kapcsolódó elemnek az előjelét megváltoztatjuk.

Mikor arra kerül a sor, hogy közöljük a játékossal, hogy mi történt, Close nem tesz különbséget kapuk és rácsok között, mint Open tette. Helyette a 17-es általános üzenet közli: „BECSAPÓDIK ÉS A ZÁR BEKATTAN".

Egy érdekes záró megjegyzés kívánkozik ide, ami az Open és Close kezelők között fennáll. Open-nek szüksége van kulcsra, Close-nak pedig — amint láttuk — nincs. Ez azt jelenti, hogy fennáll a lehetősége annak, hogy egy szegény, félrevezetett játékos a nyitott kapun át besétál egy olyan helyiségbe, amelyiknek csak egy kijárata van, és becsapja maga mögött a kaput úgy, hogy nincs nála kulcs. A 6-os és a 11-es helyiségek egyaránt ilyen kelepcék — ha a játékos van olyan bolond! A 6-os helyiség azonban biztosít egy kiutat; a bűvös „AARDVARK" szó kimenekíti a játékost a szabadságba. Borzalmas lehet azonban, ha a 11-es helyiségben kell letölteni a játék hátra levő részét!

Fogd a kincset, és fuss!

Most, hogy megtárgyaltuk a helyszín megváltoztatásának azt a speciális módját, amelyik a kapukat befolyásolja, áttérhetünk a parancsoknak arra az általános csoportjára, amely a tárgyak szállítását irányítja. A dolgok felvételének és leejtésének látszólag egyszerű aktusa végül is nem olyan egyszerű. Melyek a mozgatható tárgyak? Mennyit vihet magával a kalandozó? Az ehhez hasonló kérdésekre a megfelelő kezelőnek kell választ adnia.

Két kezelő kapcsolódik a tárgyak mozgatásának feladatához. Ezek a Take és a Drop, és a nekik megfelelő VEDD és EJTSD parancsszavak, valamint az őket követő tárgyak neve hívja be őket. Két másik szó, LOPD és DOBD az első két parancsszó szinonimája.

Nézzük először Take-et! Logikailag egy olyan kezelőnek, amely tárgyak felvételét hajtja végre, a következő kérdésekre kell megfelelnie, és ennek megfelelően cselekednie:

- A kalandozó máris túl sokat hord magával?
- Nyelvtanilag helyes parancsot adott?
- Lényt akar felvenni?
- Valami mozdíthatatlant akar elcipelni?
- A tárgy nincs már a zsákjában?
- A kért tárgy nincs is, vagy egy másik helyiségben van?

Az első kérdés arra vonatkozik, hogy a kalandozó csak meghatározott mennyiségű tárgyat hordhat magával. A *Kardhalak és kincsek*-ben ezt a maximumot szigorúan mennyiségi alapon szabjuk meg. A kalandozó legföljebb öt tárgyat hordozhat, mérettől és alaktól függetlenül. Ez bizonyos tekintetben nem reális, de kezelése egyszerűbb.

Ha a kalandozó öt tárgynál többet vihetne magával, akkor minden egyes tárgyhoz tömegszámot vagy valami hasonlót kellene hozzárendelni. Ebben az esetben a Take kezelő úgy határozná meg válaszát, hogy összegezné a jelenleg

hordott tárgyak tömegszámát és valamilyen önkényesen megválasztott maximummal hasonlítaná össze. Ha valaki szeretné ezt választani, egyszerű lenne a tárgyak mátrixának fel nem használt OB(X,0) elemeihez tömegszámokat rendelni a 0 és 255 közötti értéktartományból. Ebben az esetben egy 500 körüli össztömeget tekinthetnénk értékhatárul. Az apróbb tárgyak — mint érmék és a kulcs — 50 körüli tömegszámot kapnának, a nehéz tárgyak — mint egy arany kocka — 100-at is érhetnének. A 10. fejezetben mutatunk egy példát erre a módszerre.

Az Olvasó joggal kérdezheti, hogy egyáltalán mi a célja a hordozható tárgyak korlátozásának. Az elsődleges ok, hogy rákényszerítsük a kalandozót, hogy többször sikeresen ereszkedjen le a föld alá az összes kincs kihozásáért. Ha bármit és bármennyit magával vihetne, akkor egy hosszú kirándulás alatt mindent begyűjtene, és visszatérve a támaszpontra, befejezné a játékot. Így, hogy létezik egy felső határ, akár a mennyiség, akár a tömeg alapján, mindig végig kell küzdeni e a visszafelé vezető utat — és ezzel növeljük a kihívást.

Akárhogy is van, a *Kardhalak és kincsek* jelenlegi változata az öt tárgyat felső határként kezeli. A CT(2) változó arra a célra van fenntartva, hogy számontartsa a kalandozónál levő dolgok számát. A Take kezelőnek ellenőriznie kell, hogy CT(2) elérte-e már az ötös felső határt.

A 6-5. ábra a Take kezelő. Az első kérdésre úgy kapunk választ, hogy a CT(2)-t összehasonlítjuk öttel. Ha CT(2) már eléri vagy meghaladja a maximumot, Take nem hajlandó fölvenni a kért tárgyat. A játékost úgy értesíti a visszautasításról, hogy a B változót 36-ra állítja és meghívja a Mesprt szubrutint. Ez a 36-os üzenetet írja ki, hogy „KARJAI TELE VANNAK ... NEM BÍR EL TÖBBET". Azonban, ha CT(5)-nél kisebb, akkor Take továbbhalad, hogy szemügyre vegye a többi kérdést.

A következő kérdés nyelvtani vonatkozású. A parancs formája „VEDD FEL X-T", ahol X egy szó. A kezelő megpróbálja felismerni az X-et, hogy megtudja, melyik tárgyról van szó. Meg kell tehát próbálni a szót megkeresni a szótáblázatban, hogy megtalálja a szótárban.

A nyelvtani probléma a következő: mi történjen, ha a parancs második szava benne van a szavak táblázatában, de nem tárgy? Pl. a játékos leírhatja: „VEDD FEL NYISD". A NYISD szó benne van a szótáblázatban — de ige, és nem tárgy. A kezelőnek nem szabad megengednie egy ilyen nyelvtanilag lehetetlen esetet!

Szerencsére a program választani tud létező tárgyak és igék között. A szótáblázatban természetesen minden szóhoz tartozik egy AZ szám. Az AZ szám 5. számjegye egyes, ha a hozzá tartozó szó ige. Tehát minden szó, amelynek AZ száma 10 000 vagy annál több, ige. Ezért, mikor a Take kezelő megtalálja a parancs második szavának AZ számát, összehasonlítja 10 000-rel.

Take az Idword szubrutint használja arra, hogy az AZ számhoz hozzájusson. A parancs második szava TX$(3)-ban van. Ha A$-t egyenlővé tesszük TX$(3)-mal, és meghívjuk Idword-öt, az N változóba kerül az AZ szám értéke. Ha egy szó nem található a szavak táblázatában, N értéke nulla lesz.

```
NÉV:            TAKE
TIPUS:          KEZELŐ
FELADAT:        Tárgyak felvétele

240 IFCT(2)>=5THENB=36:GOSUB1100:GOTO104:EL
SEA$=TX$(3):GOSUB1080:IFN>9999THENB=7:GOTO2
42:ELSEIFN>12ANDN<17ORN=18THENB=40:GOTO242
241 IFN=17THENB=8:GOTO242:ELSEIFOB(N,1)=21T
HENB=9:GOTO242:ELSEIFOB(N,1)<>CT(0)ORN=0THE
NB=12:GOTO242:ELSEOB(N,1)=21:B=11:CT(2)=CT(
2)+1
242 GOSUB1100:GOTO104
```

6-5. ábra. A Take kezelő

A kezelő összehasonlítást végez 9999-cel. Ha nagyobb, a játékos nyelvtanilag hibás parancsot adott. Eredményül megjelenik a 7-es üzenet, amelyik így szól: „MIT MOND?" Ha N 10 000-nél kisebb, akkor a parancs nyelvtanilag legalább helyes, azt azonban még el kell dönteni, vajon végrehajtható-e.

A harmadik kérdést a „vájtfülű" kalandozók miatt kell feltenni. Szinte biztosra vehető, hogy valaki megpróbál fölvenni és elszállítani egy lényt. Mielőtt ezt a szempontot bevettem volna, dolgom volt egy olyan próbajátékossal, aki nem tudott átjutni az óriás ájtatos manón. Mit tett tehát? Fogta és kivitte az ostoba lényt a helyiségből! Hosszú és hangos morgolódás után beszúrtam ezt a harmadik kérdést.

Természetesen kétféle lény van: a passzív őrt álló lények, és a sokkal veszélyesebb harcias lény (a Kardhal). A passzív lényeknek 13 és 16 közé esik a tárgyszáma. A Kardhalnak, bár a helyzetét megadó információ OB(0,1)-ben megvan, a szótáblázatban található AZ számban 18-as tárgy száma van. Mikor a Take kezelő megtalálja az elszállítandó tárgy AZ számát, ezt a számot össze kell hasonlítania a lények számaival.

Ha N, az AZ szám 12-nél nagyobb és 17-nél kisebb, akkor egy passzív lényt akartak elvinni. Ha N egyenlő 18-cal, akkor a Kardhalat akarták elvinni. A parancs végrehajtása mindkét esetben meg lesz tagadva, oly módon, hogy Mesprt-t meghívjuk a 40-es üzenettel, amely így szól: „CSUDA ÖNGYILKOS HAJLAMOKAT MUTAT, APUSKÁM!" Ez mindenkit visszatart attól, hogy elvonszolja sárkányainkat!

A negyedik kérdés a mozdíthatatlan tárgyakhoz kapcsolódik. Minden helyiségnek van egy többé-kevésbé részletes leírása, amely közli a helyiség jellemzőit, színét és így tovább. Néha ez a kiírás említést tesz olyan dolgok jelenlétéről,

amelyek nem tárgyak. Pl. figyeljük meg, hogy a 14-es helyiség leírásában szerepel a következő mondat: „PÓKHÁLÓK VANNAK MINDENÜTT"! Most tegyük fel, hogy a játékos ezt a parancsot adja: „VEDD FEL A PÓKHÁLÓT"! Mivel a pókháló nem olyan tárgy, amihez egy elem tartozik a tárgyak mátrixában, mi történik egy ilyen parancsra? Ha a PÓKHÁLÓ szót teljesen kihagynánk a szavak táblázatából, akkor a parancsra ezt a választ kapnánk: „NEM LÁTOK ITT SEMMI HASONLÓT", ami nevetségesen hangzana, hisz a helyiség leírásában éppen ott vannak. De ha mégis felvesszük a PÓKHÁLÓ szót a szótáblázatba, milyen AZ számot kapjon? Feltehetően ki lehetne bővíteni a tárgyak állapotmátrixát oly módon, hogy lefedje ezeket a leíró jellegű tárgyakat, ez azonban merő pocsékolás.

Egyszerűsíti a helyzetet, ha az összes leíró jellegű tárgyat felvesszük a szótáblázatba. De ahelyett, hogy egyedi AZ számokat kapnánk, egységesen a 17-es AZ számot rendeljük hozzájuk. A kalandprogram tudni fogja, hogyan kezelje a 17-es tárgyakat — felismerhetőek, de nem valódi tárgyak.

A Take kezelő utánanéz, hogy a karnyújtásnyira levő tárgy 17-es tárgy-e. Ha az, a parancsot megtagadja. Sajnos logikailag kevés létjogosultsága van a parancs megtagadásának. Ha egyszer ott vannak a pókhálók, miért nem veheti el a kalandozó? Így ahelyett, hogy valódi magyarázatba bocsátkoznánk, meghívjuk Mesprt-t, hogy a 8-as üzenetet írja ki, amelyik a lényeget megkerüli és sziklaszilárdan kijelenti: „SIKERTELENÜL PRÓBÁLKOZIK ... NEM MOZDÍTHATÓ!" Bár elismerjük, hogy ez közel sem kielégítő, de ennél csak az lenne egyszerűbb megoldás, ha olyan helyiségleírásokat használnánk, amely a valódi tárgyakon kívül még csak nem is céloz más berendezésre vagy dolgokra. Így viszont unalmas helyszínek lennének.

A következő kérdés azt ellenőrzi, hogy egyáltalán szükséges-e a parancs. Esetleg már ott van a kalandozónál a tárgy, és nem szükséges fölvennie! Honnan tudhatjuk ezt? A játékosnál levő összes tárgy a 21-es helyiségszámot kapja. Azaz többé már nincs abban a helyiségben, ahol eredetileg állt, hanem a 21-es helyiségben, a játékos hátizsákjában tartózkodik. Ha már a kalandozónál van a tárgy, ezt a Take kezelő a tárgyak állapotmátrixának ellenőrzésével deríti ki.

Mivel az N változó megadja a tárgysorszámot, az OB(N,1) elem tartalmazza a tárgy fizikai helyét. Amikor OB(N,1) 21-gyel egyenlő, a parancs végrehajtását a program megtagadja, és a 9-es üzenetet írja ki: „MÁR ÖNNÉL VAN!"

Az utolsó kérdés, hogy a kért tárgy tényleg felvehető-e. Két különböző eset van. Az egyik esetben a tárgy esetleg egy egészen más helyiségben van. A másik esetben pedig nem található a szótáblázatában. Mindkettővel azonos módon bánunk — a kezelő azt válaszolja, hogy nem látja a dolgot a közelben.

Akárcsak az előző kérdésnél, N a tárgy számával egyenlő, OB(N,1) megadja a helyét. CT(0) megmondja, melyik helyiségben van a kalandozó. Azaz, ha OB(N,1) nem egyenlő CT(0)-lal, a tárgy egyszerűen nincs ott. Másrészt, ha a játékos a program szótárából hiányzó tárgyat akar fölvenni (pl. „VEDD

FEL A KOALÁT"), akkor az N változó értéke nulla, mert ez az eredmény, ha az Idword szubrutin nem talál valamit a szótáblázatában. Ha pedig N nullával egyenlő, vagy a másik eset fordul elő, a parancsot a 12-es üzenettel tagadja meg a program, amelyik így szól: „NEM LÁTOK ITT SEMMI HASONLÓT". Figyeljük meg, hogy ez a beszédmód nem fedi fel, hogy a program nem ismeri a parancsban említett dolgot, a játékos máshol esetleg *találhat* Koalát!

Ha a Take átjut a fenti hat lehetőségen, készen áll feladata elvégzésére, amit három lépésben végez el. Először a tárgyat a játékos birtokába kell juttatnia. Ehhez eltávolítja a helyiségéből és a hátizsákba teszi. Az OB(N,1) változóba 21 kerül, ez intézi el az átvitelt. Másodszor a programnak követnie kell, hány dolgot visz a kalandozó. Ezt Take úgy oldja meg, hogy CT(2) aktuális értékét eggyel növeli, amivel nyilvántartja a teljes leltárt. Végül a játékost értesíteni kell az akció sikeréről. Ebből a célból a 11-es „RENDBEN" üzenet jelenik meg. Szokás szerint félrevezető rövidséggel, elhomályosítva azt a bonyolult döntéshozatalt, ami idáig vezetett!

Elintéztük a tárgyak felvételét. Most vizsgáljuk meg, hogyan ejtünk el egy tárgyat valamelyik helyiségben!

Dobd el a kincset!

A szótáblázatban két szinonimaként kezelt kulcsszó található, ami a szállított tárgyak lerakásához kapcsolódik: EJTSD és DOBD. Mindkettő a 6-6. ábrán látható Drop kezelőt hívja meg.

Drop működése hasonló Take-éhez, de annál egyszerűbb. A kezelő három kérdésre keres választ, mielőtt a parancsot végrehajtaná:

- Helyes-e nyelvtanilag a parancs?
- Ott van-e a hátizsákban a tárgy?
- A kalandozó netán az Elvarázsolt gránátot akarja eldobni?

Mindhárom kérdés a lerakni kívánt dolog, tárgy sorszámától függ. Ezért meghívjuk Idword-öt, hogy keresse meg a TX$(3)-ban tárolt szót valahol a szótáblázatában, és a szó AZ számát tegye az N változóba. (Tárgyak esetében az AZ szám egyenlő a tárgy számával.)

Az első kérdést ugyanúgy intézzük el, mint a Take kezelőben. Ha a játékos igét használt az EJTSD parancs tárgyaként, akkor N értéke nagyobb, mint 9999, azaz 10 000 vagy annál több, mivel igéknél az 5. számjegy értéke egy. Ebben az esetben a 7-es üzenet még egy „dobást" ad a játékosnak, közölvén: „MIT MOND?"

A következő kérdést hasonló módon intézzük el, mint Take-nél, de ellentétes eredménnyel. Ebben az esetben a parancsot akkor tagadja meg a program, ha a tárgy nincs a játékos birtokában. Ha OB(N,1) nem egyenlő 21-gyel, a tárgy nincs a hátizsákban. Mesprt a 10-es üzenetet írja ki: „NINCS ÖNNÉL!"

Az utolsó kérdés teljes tisztázásával meg kell várni a következő fejezetet; van egy tárgy, az Elvarázsolt gránát, amelyik igen furcsán viselkedik, ha meg-

```
NÉV:          DROP
TIPUS:        KEZELO
FELADAT:      Tárgyak lerakása

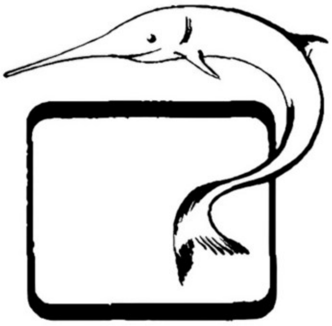
260 A$=TX$(3):GOSUB1080:IFN>9999THENB=7:GOS
UB262:ELSEIFOB(N,1)<>21THENB=10:GOTO262:ELS
EIFN=12THEN540:ELSEOB(N,1)=CT(0):B=11:CT(2)
=CT(2)-1
262 GOSUB1100:GOTO164
```

6. ábra. A Drop kezelő

róbálják elejteni vagy eldobni. A tárgy száma 12; ha a Drop kezelő úgy találja, ogy a 12-es tárgyat akarják eldobni, az egész ügyet az 540-es sor hatáskörébe talja, ez a Bomb kezelő kezdete. Látni fogjuk, hogy egy sor különleges dolog ırténhet, ha Bomb meghívására kerül sor, ám ez egy egészen más történet.

Most, hogy áttekintettük a legszükségesebbeket, megtörténhet a lerakás. .kár az előbb, most is három lépés van. Megváltozik a tárgy helyzete, oly ódon, hogy OB(N.1) egyenlő lesz a helyiség CT(0)-ban tárolt számával. . továbbcipelt tárgyak számát frissíteni kell: egyet kivonunk CT(2)-ből. Végül ıegjelenik a „rendben" üzenet.

A kalandozó fokról fokra halad előre. Néhány fejezettel korábban nem ıdott mást, mint sétálni és nézelődni. Most már képes felkapni és elvinni a .rgyakat, kapukat tud nyitni és zárni. A következő fejezetben a kalandozó .egtanulja, hogyan védje meg magát a barlangok sötét, föld alatti folyosóin :abadon kószáló lényekkel szemben.



7. FEJEZET

Küzdelem az ellenféllel

A veszélyesség az, ami a kalandprogramot a húsvétitojás-kereséstől megkülönbözteti. Ha a kalandozónak nincs más dolga, mint kóborolni és kincsekre lelni, akkor nincs kihívás! Kell lenni valaminek, ami ellenszegül kísérleteinek, valaminek, ami gátolja előrehaladását, sőt az életét veszélyezteti. Ezért vannak különböző lények a kalandprogramokban.

Más-más programok eltérően kezelik lényeiket. Egyes lények céltalanul kóborolnak a helyszínen, véletlenszerűen ütköznek bele a kalandozóba. Egyeseknek állandó őrhelyük van, amit folyamatosan strázsálnak. Egyesek nem támadnak, amíg nem fenyegetik őket. Másokat nem lehet közönséges fegyverrel elpusztítani. A küzdelem kimenetét meghatározhatják teljesen véletlen tényezők, vagy számon lehet tartani a küzdők erejét, és ennek segítségével eldönteni, kit illet meg igazság szerint a győzelem.

A küzdelem szimulációja során a *Kardhalak és kincsek* a fenti változatok kombinációját használja. Kísérletet tettünk arra, hogy az algoritmusok legyenek egyszerűek, de fenntartsuk a tényleges harc illúzióját. A programban háromféle küzdelem van:

- Támadás/megtorlás egyes passzív lények esetében.
- Különleges fegyverek más passzív lények ellen.
- Védekezés/támadás a kitartó lénnyel szemben.

A 7-1. ábrán látható a színfalak mögött várakozó lények listája. Emlékezzünk rá, hogy egy kalandprogramban igazában alapvetően kétfajta lény van! Az egyiket passzív lénynek nevezhetjük. Fő feladata, hogy a helyszín bizonyos átjáróit őrizze vagy lezárja. Mint ilyen, egyben tisztességes akadály, és benne van az akadályok táblázatában. Más akadályok, pl. a kapuk, a „NYISD" parancs hatására válnak járhatóvá. A passzív akadályokat harccal küzdhetjük le. Saját kezdeményezésükből nem támadnak, de mindig szembeszállnak, ha támadás éri őket. Mivel nem válnak közvetlenül ellenségessé, nem szükségszerű, hogy a kalandozó harcba bocsátkozzék velük. A játékos viszont pontot kap

TIPUS	LÉNY	FEGYVER	HELYISÉG
PASSZIV	ÓRIÁS ÁJTATOS MANÓ	SZEKERCE	4
PASSZIV	ÓRIÁSGYIK	SZEKERCE	18
PASSZIV	FEHÉR PÓK	GRÁNÁT	14
PASSZIV	NEVESINCS BORZALOM	GRÁNÁT	6
AKTIV	DÜHÖS KARDHAL	SZEKERCE	

7-1. ábra. A KARDHALAK ÉS KINCSEK lényei

minden megölt lényért, és vannak bizonyos kincsek, amelyekhez nem juthat hozzá anélkül, hogy ne küzdene le egy passzív lényt.

További kihívást jelent, hogy a passzív lényeket nem lehet azonos módon legyőzni. A négy passzív lény ebből a szempontból két kettes csoportra osztható. Az egyik csoportot a támadás/visszavágás ciklussal lehet megtámadni és végül megölni. A másik csoport teljesen érzéketlen a szokásos fegyverrel (a szekercével) szemben, sőt még visszavág. Ezt a két lényt csak egyfajta módszerrel lehet megölni, az Elvarázsolt gránáttal.

A passzív lényektől elválik a sokkal veszélyesebb kitartó lény, a Kardhal. A harc 3. osztályát védekezés/támadásnak nevezzük, mivel a Kardhal provokálás nélkül támad. A szokásos módon (vagyis a szekercével) lehet megölni, de helyiségről helyiségre követi a kalandozót. A játékos csak két módon menekülhet meg: vagy megöli a Kardhalat, vagy felszalad a föld színére, ahová a Kardhal nem tudja követni.

A kód többi része is közreműködik a harc szimulálásának támogatásában. Közülük egyet már ismertettünk néhány fejezettel korábban a Vezérlő vizsgálata során. Az a rész a kitartó lény mozgását irányítja, azt, hogy vajon támad-e és mennyire sikeresen. A szabványos csatakezelőt Fight*-nak hívjuk, és ez szabja meg az összes olyan lénnyel történt találkozás kimenetelét, amelyik szekercével elpusztítható. A Bomb nevű kezelő szabályozza az Elvarázsolt gránát hatását annál a két lénynél, amelyiknek túl vastag a bőre és ellenáll a szekercének. Végül ide tartozik a Resur nevű kezelő, ami akkor lép életbe, amikor a játékos meghal. Újjáéleszti a játékost, és kirakja a barlangból, módosítja a pontozást, és ellát még néhány részfeladatot.

*Fight=harc.

Vívjunk vérbeli csatát!

A szótáblázatában három szó kapcsolódik a szokásos csatához. Ezek: ÖLD, KÜZDJ és PUSZTÍTSD. Mindhármat szinonimaként kezeljük, és ugyanazt a szubrutint hívják meg: a Fight nevű 7-es számú kezelőt. Kódja a 7.2. ábrán látható.

A Fight kezelő a következő kérdésekre válaszol, ha meghívják:

- Jelen van a kitartó lény?
- Jelen van egy passzív lény?
- Kéznél van a szokásos fegyver?
- Érzéketlen-e a lény a szokásos fegyverrel szemben?

Feltéve, hogy a csatára sor kerül, Fight a következő módon beállított valószínűségeknek megfelelően mérsékeli a csetepatét:

- 70 százalék esélye van annak, hogy a lény elpusztul ebben a menetben.
- Ha nem a kitartó lényről van szó, 30 százalék az esélye annak, hogy a visszavágás során a kalandozó meghal.

Jegyezzük meg, hogy a 30 százalékos visszavágási szám csak passzív lényekre érvényes! A Fight kezelő nem váltja ki a kitartó Kardhal visszavágását. Ehelyett a Vezérlő irányítja a Kardhal reakcióját.

Először arra a kérdésre kell felelni, vajon a kihívott ellenfél a Kardhal-e. Emlékezzünk rá, hogy a Vezérlő tárgyalásánál elmondtuk, hogy a tárgyak állapotmátrixának fel nem használt OB(0,0) és OB(0,1) eleme a kitartó lény számára van fönntartva! OB(0,1) adja meg a helyzetét, és a Vezérlő véletlenszerűen mozgatja fel-alá. Míg a játékos és a Kardhal nincs ugyanabban a helyiségben, OB(0,0) értéke nulla. Ha azonban a Kardhal a játékos körül ólálkodik, OB(0,0) értéke egyre lesz beállítva. Ettől kezdve, míg a játékos meg nem öli a Kardhalat vagy a felszínre nem menekül, a Kardhal helyiségről helyiségre követi a játékost.

Ebből következik, hogy OB(0,0) segítségével könnyen megállapítható, hogy a Kardhal a közelben van-e. Ha eggyel egyenlő, Fight automatikusan feltételezi, hogy a játékos megpróbálja elpusztítani a Kardhalat. A kezelő továbblép a következő sorra, hogy a többi kérdést elintézze.

Ez egy érdekes szempontot vet fel. Ha a játékos leírja az ÖLD parancsot, és a helyiségben két lény van, akkor Fight először mindig a kitartó Kardhalat tételezi föl. Ezért a játékosnak nem kell megadnia a lény nevét a csata hevében, és a kezelő nem jön zavarba attól, hogy egyszerre kétfajta lénnyel van dolga. Ezt a feltételezést nem túl nehéz elfogadni, mivel a Kardhalat nehéz figyelmen kívül hagyni, hisz csak egy bolond vesztegeti az idejét azzal, hogy egy álmos, passzív lényt provokáljon, miközben a Kardhal a torkának ugrik.

A következő kérdés, hogy van-e más lény a helyiségben, amit úgy ellenőrzünk, hogy lefuttatunk egy POR-NEXT ciklust, amelyik végigpásztázza a tárgyak állapotmátrixát. A passzív lények olyan tárgyak, amelyekhez 13 és 16 közti szám tartozik. A négy lény mindegyikének helyzetét összehasonlítjuk

```
NÉV:            FIGHT
TIPUS:          KEZELŐ
FELADATA:       Harc a lényekkel

320 IFOB(0,0)=1THEN322ELSEFORK=13TO16:IFOB(
K,1)<>CT(0)THENNEXTK:B=41:GOSUB1100:GOTO104

322 IFOB(10,1)<>21THENB=23:GOTO326:ELSEIFK=
15ORK=16THENB=24:GOTO324:ELSEX=RND(100):IFO
B(0,0)=1THEN328ELSEIFX>70THENB=26:GOTO324:E
LSEOB(K,1)=0:A=1:GOSUB1200:GOSUB1220:B=25:G
OTO326
324 GOSUB1100:B=27:GOSUB1100:X=RND(100):IFX
<40THENB=29:GOSUB1100:GOTO580:ELSEB=28
326 GOSUB1100:GOTO105
328 IFX>70THENB=26:GOSUB1100:GOTO112:ELSEOB
(0,0)=0:OB(0,1)=0:B=25:CT(4)=CT(4)+25:GOTO3
26
```

7-2. ábra. A Fight kezelő

CT(0)-val, a játékos helyzetével. A ciklus addig folytatódik, amíg egyezést nem talál. Ha egyezés fordul elő, a vezérlés átadódik a következő sorra, további vizsgálatok céljából. Ha nem fordul elő egyezés, vagyis nincs jelen passzív lény, a ciklus végig lefut. A kezelő csak azt tételezheti fel, hogy szegény, eszelős játékos egy sziklát vagy valami hasonlót próbált megtámadni és elpusztítani. A B változóba 41-et tölt, és meghívja a Mesprt szubrutint, hogy írja ki a következő üzenetet: „CSAK SEMMI PÁNIK! NINCS ITT SEMMI VESZÉLY!"

Hozzátehetjük, hogy a Fight-ot azért kellett ezzel a résszel kiegészíteni, hogy elrejtsünk egy zavaró helyzetet. Egy próbajátékos találta meg ezt a hézagot. Játék közben belépett egy olyan helyiségbe, amelyben semmilyen lény sem volt, kiadta a KÜZDJ parancsot, és hirtelen megjelent egy nem létező lény, nekiugrott a torkának és megölte! Programozók, vigyázat! Ha nem gondoltatok át minden lehetőséget, a játékos jó pár szarvashibára bukkan majd rá!

A harmadik kérdés arra vonatkozik, vajon van-e a játékosnál fegyver, amivel harcolhat! A szokásos fegyver a 10-es tárgy, a szekerce. Ha a játékos hátizsákjában van, akkor OB(10,1) egyenlő 21-gyel, a birtoklást jelző számmal. Ha nem, akkor a játékos fegyvertelen, és a kezelő a 23-as üzenettel válaszol, ami így szól: „MILYEN FEGYVERREL?" Figyeljük meg, hogy ez a kérdés nem teszi lehetővé, hogy ennek a kezelőnek a segítségével használjuk az Elvarázsolt gránátot. Ha a játékos bombázni akarja ellenfelét, akkor a kifejezetten erre való BOMBÁZD parancs kulcsszót kell leírnia.

Még akkor is, ha a játékosnak van szekercéje, van két lény, amelyik túl kemény ahhoz, hogy ártana neki. Ez a Fehér pók és a Nevenincs borzalom, tárgyszámuk 15 és 16. Ha a kalandozó bármelyik lényre sújt le szekercéjével, a kezelő a 24-es üzenetet írja ki: „A SZEKERCE CSAPÁSAI ERŐTELJESEK ... DE HATÁSTALANOK!" A lénynek nem esik baja, de a kezelő áttér a következő sorra, amely a visszavágást irányítja. Így a játékos belehalhat abba, hogy megtudja a titkot, miszerint a szekerce nem öli meg ezt a két bestiát.

Ha túljutottunk a négy előzetes akadályon, elkezdhetjük a játékos támadásának szimulálását. Az X változóba egy 0 és 100 közötti véletlen érték kerül. Ez a szám jelenti a támadás sikerét vagy kudarcát jelentő valószínűségi százalékot. Mielőtt kiértékelné ezt a valószínűséget, a program kétfelé ágazik el. Ha a lény kitartó, akkor végzetét vagy túlélését másként kell kezelni, mint a passzív lényét. Erről a 328-as sor gondoskodik, amint azonnal látni fogjuk.

Passzív ellenfélnél a véletlen százalékot vizsgáljuk. Ha X 70-nél több (30 százalékos esély), akkor a támadás sikertelen volt, a lény túlélte. Ebben az esetben a 26-os üzenet jelenik meg: „ELHIBÁZTA! PFUJ". A vezérlés a következő sorra kerül, és a lény kap egy esélyt a visszavágásra.

Mi történik, ha X legfeljebb 70? Ha így áll a dolog, a szekerce célba talált, a kezelőnek el kell távolítania a lényt. Ehhez szükség van néhány lépésre. Először is, a lényt el kell távolítani a helyiségből, hogy a Vezérlőleíró alárendelt része kihagyja a leírásból. Ennek leghatásosabb módja, hogy a nem létező 0-s helyiségbe tesszük át. A lény helyzetét OB(K,1) mutatja, mivel passzív lény esetén K egyenlő a lény tárgyszámával, a 320-as sorban található FOR-NEXT ciklus eredményeként. Ha OB(K,1)-be nullát töltünk, ez mintegy a feledés homályába küldi a lényt.

Ez még nem minden. A passzív lény nem pusztán egy tárgy. Egyben akadály is, amelyhez egy elem tartozik az akadálylistában. A Fight kezelőnek úgy kell megváltoztatni ezt az elemet az akadálylistán, hogy a játékos szabadon járhasson át azon az átjárón, amit eddig a lény őrzött. Ezt két szubrutin segítségével végezzük el. Az első, a Ckobs megkeresi a lényhez tartozó elemet az akadálylistában. A második, Revobs átállítja az elem állapotát járhatatlanról járhatóra. (Ezeket tisztáztuk az előző fejezetben.)

Az 1200-as sorban található Ckobs szubrutinnak ismernie kell az akadály típusát, és hogy jelenleg hányas számú helyiségben található, ahhoz, hogy az elemet megtalálja. A helyiség száma mindig CT(0)-ban van; az akadály típusát az A változóban kell tárolni. A kezelő 1-et tölt A-ba (lény típusú akadály), és meghívja Ckobs-t. Mikor a szubrutin lefut, A-ba kerül az elem száma, vagyis egy 1 és 10 közé eső szám.

Az 1220-as sorban található Revobs szubrutinnak ahhoz, hogy az elemet megtalálja, ismernie kell az akadálylistán belüli sorszámot. Elvárja, hogy ez a szám az A változóban legyen. Szerencsére Ckobs A-ba tette ezt a számot, tehát nincs szükség további előkészítésre. Revobs közvetlenül Ckobs után meghívható. Mire a szubrutin lefut, az akadálylistán az elem már jelzi, hogy az átjárón át szabad az út.

A passzív lény elpusztításának befejező lépése a gyászjelentés. A Mesprt szubrutin a 25-ös üzenetet írja ki, amely így hangzik: „A BŰVÖS SZEKERCE

LESÚJT! A LÉNY ELTŰNIK EGY BŰZÖS FÜSTFELHŐBEN!" Ezzel a kezelő befejeződik, és visszatér a Vezérlőparancs alárendelt részébe.

Az utolsó néhány bekezdés a passzív lény megtámadásával foglalkozott. Mielőtt a lény visszavágását néznénk meg, lássuk, mi történik, ha az ellenfél a kitartó lény. Ebben az esetben a 328-as sor kezeli a támadást. Az X változóban még mindig egy 0 és 100 közötti véletlen szám van. Ha X 70-nél nagyobb (30 százalékos esély), akkor a Kardhal kikerülte a játékos szekercéjét. Megjelenik a 26-os üzenet („ELHIBÁZTA! PFUJ!"), de ahelyett, hogy a visszavágást kezelő programrész kerülne sorra, a vezérlés visszaadódik a Vezérlőbe, közvetlenül a Kardhal leírását megelőző részre. A Vezérlőnek ez a része váltja ki, hogy a Kardhal maga lendüljön támadásba. Azzal, hogy így bontottuk fel a dolgokat, lehetőséget teremtettünk rá, hogy a Kardhal ismételten támadjon, fáradhatatlanul, esetleg minden menetben, és így a legnehezebben legyőzhető lénnyé váljon.

Ha az X-ben található valószínűségi érték legfeljebb 70, a Kardhalat elérte a végzete. Figyeljük meg azonban, hogy kimúlását egyedi módon kezeljük.

Az OB(0,1) változó szabja meg, hogy hol van; ezt nullára állítjuk be, így a nem létező helyiségbe küldjük. Az OB(0,0) változó irányítja cselekedeteit. Ezt nullára állítjuk, ezzel várakozó állapotba helyezzük. Röviden, a Kardhal sohasem hal meg igazán. Épp csak ideiglenesen a 0-s helyiségbe kerül. A Vezérlő véletlenszerű alapon ismét útnak indítja. Így, a kalandozó útja során hamarosan ismét találkozik a Kardhallal. Ezt a jellegzetességet leginkább úgy lehet elképzelni, hogy a barlangi tavak hemzsegnek a Kardhalaktól, és mindig egy újabbal találkozunk. Ez a véletlenszerűen ismétlődő veszély növeli a játék érdekességét.

Habár a Kardhal (ha úgy tetszik egy Kardhal) visszatér, a haláleseményt bejelentő üzenet azonos azzal, amely az örökre eltűnő passzív lényeknél használatos; a 25-ös üzenet.

Minden egyes Kardhal megölése után a CT(4) változó 25-tel nő. Ezt a következő fejezetben fejtjük ki részletesen, most elégedjünk meg annyival, hogy ez pontozási célokat szolgál! A játékos 25 jutalompontot kap minden egyes elpusztított Kardhalért.

Ezzel elintézzük a Fight kezelő támadási részét. A 324-es sor jelenti a visszavágási kísérletet. A Mesprt szubrutint hívó GOSUB-bal kezdődik, mivel a kezelő többi része kiírandó üzenetekkel ér ehhez a sorhoz. Külön üzenetet ír ki, a 27-est, amely így kiált: „A FÖRTELMES SZÖRNY A TORKÁNAK UGROTT!"

Ezután kerülnek kiszámításra a lény visszavágásának sikerét vagy sikertelenségét megszabó valószínűségek. Mint korábban, az X változóba 0 és 100 közötti érték kerül. Ha X 30-nál kisebb (30 százalékos esély), akkor a lény győzedelmeskedik és a kalandozó elpusztul. Ebben az esetben elővesszük és kiíratjuk a 29-es üzenetet, amely fájdalmasan panaszolja: „VÉGZETT ÖNNEL!!" Ezen a ponton a vezérlés az 580-as sorra kerül, a Resur szubrutinra. Ez a kezelő (vagy alárendelt kezelő) intézi a játékos újbóli elindítását.

70 százalékos esélye van annak, hogy a lény visszavágása sikertelen. Ebben az esetben a 28-as üzenet jelenik meg, amely ideget borzolva közli: „VALAHOGY SIKERÜLT ELHÁRÍTANI!!” Minden a legnagyobb rendben, a vezérlés ismét a Vezérlőre kerül, a játékos lélegzetvételnyi szünethez jut, mielőtt egy újabb „ÖLD” vagy „PUSZTÍTSD” parancsot adna ki.

A játékos felélesztése

Egyszer egy évben a legtapasztaltabb játékost is felfalják. Azért, hogy kaphasson egy második lehetőséget, a Resur kezelő visszahozza egy újabb fordulóra. Ennek persze hatása van a pontokra. (Így arra kényszeríti a játékosokat, hogy inkább leleményesek legyenek, és ne túl vakmerőek.)

A Resur a 7-3. ábrán látható. Egy halom feladatot kell ellátnia; ezek a következők:

- Tartsa számon pontozás céljából, hogy hányszor halt meg a játékos.
- Tudósítsa a játékost helyzetéről.
- Biztosítsa, hogy a játékosnak legyen fáklyája a következő vállalkozáshoz.
- Ürítse ki a hátizsákot abban a helyiségben, ahol a játékos meghalt.
- Vigye vissza a játékost a támaszpontjára.
- Állítsa nullára a nála levő tárgyak számát.

A Resur azzal kezdi, hogy feljegyzi a halált a CT(3) változóban. Később, a pontszám kiszámításánál, CT(3)-at is figyelembe vesszük, és az összpontszám halálonként 20 ponttal csökken.

Ezután kiíratjuk a 35-ös üzenetet, hogy tudassuk a játékossal a szörnyű helyzetet. Ez így hangzik: „NOS, DICSŐ KALANDOZÓ! IGAZÁN NAGY BAJBAN VAN! SZERENCSÉRE MÓDUNKBAN ÁLL, HOGY ÚJRA-

```
NÉV:            RESUR
TIPUS:          KEZELŐ
FELADATA:       A megölt játékos fel-
                támasztása

580 CT(3)=CT(3)+1:B=35:GOSUB1100:OB(9,1)=2:
FORI=1TO12:IFOB(I,1)=21THENOB(I,1)=CT(0):NE
XT:ELSENEXT
582 CT(0)=1:CT(2)=0:GOTO100
```

7.3. ábra. A Resur kezelő

ÉLESSZÜK ÖNT!... PUFF!!!" Ezen a ponton kívánatos valamilyen FOR-NEXT ciklust beiktatni a kezelőbe, egyszerűen csak az időhúzás kedvéért. Így azt az illúziót keltjük, hogy micsoda erőfeszítésbe kerül a darabokra szakadt kalandozót ismét összerakni.

A következő tennivaló az, hogy ha már egyszer újjáélesztettük a játékost, akkor ismét képes legyen leereszkedni a föld alatti barlangokba. Amikor visszatér, kívül van a föld felszínén. Szükség van tehát egy fáklyára a föld alatti utazáshoz — fáklyáját viszont elejtette valahol „odalenn". Marad az egyetlen sportszerű eljárás: elhelyezünk egy fáklyát a közelében, hogy visszatérhessen a föld alá, és visszaszerezze kincseit. A Resur fogja a fáklyát, a 9-es tárgyat, és a felszínen fekvő 2-es helyiségbe teszi. Itt a játékos könnyűszerrel ráakad.

Most a Resur-nek el kell lopnia mindent a játékos hátizsákjából. Igazság szerint minden tulajdona ott hullott szét a földre, ahol meghalt. Egy FOR-NEXT ciklus átvizsgálja a mozdítható tárgyak listáját (tizenkettő van belőlük; a többi lény). Minden tárgy, ahol a helyzetet jelző szám 21, a játékos birtokában van. Az összes ilyen tárgy átkerül abba a helyiségbe, ahol a játékos holtan fekszik, amit CT(0) mutat meg.

Az utolsó lépés az 582-es sorban zajlik le. Azáltal, hogy CT(0)-t 1-re állítjuk, a játékos eltűnik a barlangrendszerből és a támaszpontján pottyan le, vagyis az 1-es helyiségben, az ásatási üregben. Ezenkívül, mivel a játékos most már semmit sem hord, CT(2)-t nullára kell állítani. CT(2)-t a Take kezelő használja, hogy megállapítsa, nem hordoz-e a játékos túl sokat. Ennél a pontnál alaphelyzetbe kerül.

Az ellenfél bombázása

Meg kell vizsgálni még egy utolsó kezelőt, amelyik kapcsolatban áll a kalandozó élet-halál harcával. Emlékezzünk rá, hogy a négy lényből kettő érzéketlen a játékos szapora szekercecsapásaival szemben! Sem a Fehér pók, sem a Nevenincs borzalom nem ölhető meg szekercés támadással; azonban visszavágnak. Jaj annak a kalandozónak, aki a Jóslatok termének csapdájába esik pusztán egy szekercével! Egyetlen menekvése a bűvös utazás, mert a Nevenincs borzalom őrzi az egyetlen kijáratot, és csak kacag a szekerce láttán.

Szerencsére van egy fegyver, amely mindkét kemény bestiával végez. Ez az Elvarázsolt gránát, amely a 12-es tárgyként is ismert. Ha ezt a bűvös bombát hozzávágjuk valamelyik keményebb lényhez, az robbanásával légies fényjelenség kíséretében elsöpri a bestiát. Más tárgy ellen a gránát nem működik és nem robban; így a kalandozó rá van kényszerítve, hogy egy-két próbálkozást tegyen bezárt kapuk felrobbantására.

A 7-4. ábrán látható a Bomb (bombázd, bomba) nevű kezelő, amely az Elvarázsolt gránát működését ellenőrzi. Bomb két módon hozható működésbe. Az elsőt megemlítettük az előző fejezetben, amikor a Drop kezelőt ismertet-

```
NEV:              BOMB
TIPUS:            KEZELO
FELADATA:         A buvös granát kezelese

540 IFOB(12,1)<>21THENB=20:GOTO544:ELSEOB(1
2,1)=CT(0):CT(2)=CT(2)-1:FORK=15TO16:IFOB(K
,1)<>CT(0)THENNEXTK:B=21:GOTO544:ELSEOB(K,1
)=0:A=1:GOSUB1200:GOSUB1220:B=22
544 GOSUB1100:GOTO104
```

7-4. ábra. A Bomb kezelö

tük. Az egyik kérdés, amit Drop feltesz, így hangzik: „A gránátot ejti el a kalandozó?" Ha igen, akkor Drop az egész ügyet a Bomb hatáskörébe utalja. Mivel a Drop-ot behívó parancsnak alapvetően két alakja van, a kezdet kezdetétől két parancs áll rendelkezésre, amely Bomb-ot aktivizálja. Ezek a „DOBD EL A GRÁNÁTOT" és az „EJTSD EL A GRÁNÁTOT".

A kalandozó azonban ennél pontosabban is kifejezheti magát. A szavak táblázatában szerepel két kulcsszó, amely kifejezetten a 17-es kezelő (Bomb) aktivizálását igényli. A két szó „BOMBÁZD" és „ROBBANTSD". Mint látni fogjuk, ez esetben a kezelő teljesen figyelmen kívül hagyja a parancs 2. szavát. A játékos beírhatja, hogy „BOMBÁZD A LÉNYT" vagy „ROBBANTSD FEL A PÓKOT", vagy bármi mást, a program mégis megérti a játékos szándékát.

A Bomb kezelőnek a következő tényezőket kell meghatároznia, mielőtt továbbhaladna a gránát felrobbantása felé:

- Ott van-e a játékosnál a gránát?
- A közelben van-e a kemény lények közül valamelyik?

A Bomb azzal kezdi, hogy ellenőrzi, vajon nem blöfföl-e a játékos. Van-e eldobni való gránátja? Az Elvarázsolt gránát a 12-es tárgy. Ha a játékos birtokában van, akkor az OB(12,1) változó értéke 21, a hátizsák száma. Ha ez nem igaz, Mesprt meghívása következik, és a 20-as üzenet jelenik meg: „NINCS IS BOMBÁJA!"

Feltéve, hogy a gránát a játékosnál van, a kezelő továbbmegy és a földre dobja. Ez két lépésben történik meg. Először a gránát a hátizsákból a jelenlegi helyiségbe kerül úgy, hogy OB(1,1)-be CT(0)-t teszünk. Másodszor, mivel a játékos hátizsákja kicsit könnyebb lett, ezt a tényt fel kell jegyezni. A CT(2) változó, a leltári tárgyak száma eggyel csökken.

A következő kérdés, hogy vajon szabályos célpont van-e a hatósugáron

belül. Az a két lény, amelynek jelenléte működésbe hozza a gránátot, a Pók és a Borzalom; ezek a 15-ös és a 16-os tárgyak. Egy rövid FOR-NEXT ciklus ellenőrzi, hogy a kettő közül valamelyik a helyiségben van-e, oly módon, hogy helyzetüket összehasonlítja CT(0)-val. Ha egyik sincs a helyiségben, a gránát csütörtököt mond. A 21-es üzenet jelenik meg és közli: „A GRÁNÁT A FÖLDRE HULL, SEMMI SEM TÖRTÉNIK".

(Meg kell még jegyezni, hogy a gránát sosem „használódik el". Akár felrobban, akár nem, a földön köt ki és nyugton marad. Fel lehet venni, és később fel lehet használni a másik kemény lény ellen. Igazán nem lehet azt mondani, hogy szűkmarkúak lennénk.)

Ha a két lény egyike a helyiségben van, a kezelő megsemmisíti. Az első lépés egyszerű. Nullára állítjuk OB(K,1)-et, a lényt a nem létező 0-s helyiségbe száműzzük, ahonnan sosem tér vissza. A feladat neheze, hogy az akadálylista hozzá tartozó elemét úgy módosítsuk, hogy a lény által őrzött átjáró szabaddá váljon. Ezt pont úgy érjük el, mint a Fight kezelőben. Az A változót egyre állítjuk, ez jelzi a lény típusú akadályt, és meghívjuk a Ckobs szubrutint. A Ckobs átnézi az akadálylistát, és rátalál a megfelelő elemre. A kezelő meghívja Revobs-t, amely a tárgy állapotát járhatatlanról járhatóra változtatja.

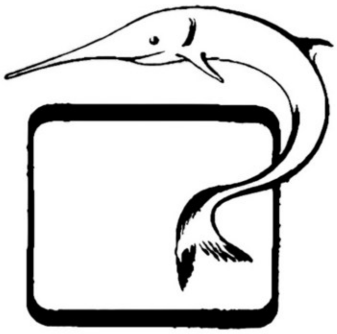
A végső feladat, tudatni a játékossal a nagy diadalt. A 22-es üzenet teszi ezt meg: „A GRÁNÁT HANGTALANUL, TERMÉSZETFÖLÖTTI KÉK LOBBANÁSSAL FELROBBAN... A LÉNYNEK VÉGE!"

A kalandozó most már mindenre készen áll! Egyik kezében szekerce, másik kezében gránát, minden bestiával szembeszáll — legyen az Kardhal, Ájtatos manó vagy Borzalom. Csak azt kell tudnia, melyik fegyver melyik lény ellen hatásos!

Épen, egészségesen

Az előző fejezetekben részletesen megbeszéltük egy kalandprogram fő alkotóelemeit. A játékos bejárhatja a helyszínt, bekukkanthat a helyiségekbe, csak hogy a látványban gyönyörködjön. A játékos hathat is környezetére, kapukat csukhat be, tárgyakat vihet el, és így tovább. Ezenkívül csatákat vívhat, bestiákat győzhet le különleges fegyverekkel, győzhet vagy néha veszíthet.

Hátra van még egy néhány parancs, amit a kalandprogram elfogad; ezek azonban másodlagosak a játék szempontjából. Pl. a játékost érdekelheti a pontszáma, esetleg meg akarja vizsgálni a hátizsák tartalmát. A következő fejezetbe ezekre a kiegészítő parancsokra kerül sor.



8. FEJEZET

Kisegítő parancsok

A kalandprogram legtöbb parancsa a játékos mozgását vagy tevékenységét szabályozza. A játékos mozog, kapukat nyit ki, dolgokat vesz fel, szörnyekkel csatázik. Jó néhány parancs azonban magának a programnak szóló utasítás, vagy azt a módot szabályozza, ahogy a játékot játsszák, vagy bizonyos információt kér. Ezek a *kisegítő parancsok.*

Hét olyan kezelő van hátra, amely a kiegészítő parancsokat támogatja. A hétből kettő a környező helyszínről ad információt. További kettő a pontszámot közli, ebből egy a játék befejezésével jár együtt. Megint másik kettő a TRS–80-hoz mellékelt magnetofont használja, hogy a játék állását eltegye egy későbbi alkalomra, és hogy a játékos hosszú időt tölthessen a barlangban. Az utolsó kezelő egysoros üzenetekkel válaszol bizonyos beírt üzenetekre, szigorúan csak a hatás kedvéért.

Lássuk csak még egyszer!

Miközben a kalandozó helyiségről helyiségre vándorol, a program számontartja, hogy melyik helyiségben járt már, és melyik az új számára. Erre az információra támaszkodva egy helyiség leírása kétféle lehet: az első alkalommal egy bekezdésnyi leírás, vagy a helyiség rövid neve az ezt követő alkalmakkor. Ez utóbbi csökkenti a szem kifáradását és az egyhangúságot, amit az örökös hosszú leírás jelentene, valahányszor a helyiségbe lépünk.

Csak egy baj van ezzel a készséges szolgáltatással: az, hogy a játékosok felejtenek. A játékos hajlamos rá, hogy besétáljon a Pókhálós terembe és elfelejtse, hogyan is festett. Hol vannak a kapuk? Vannak-e kikerülendő veszélyes sziklák?

Ezért van a kalandprogramban a MUTASD parancs. A parancs hatására ismét megjelenik a helyiség bővebb leírása, és így a bejáratok és a kijáratok tisztán láthatók.

```
NÉV:          LOOK
TIPUS:        KEZELŐ
FELADAT:      Helyiségek bövebb leirása

380 C=0:GOSUB1160:GOSUB1140:GOTO104
```

8-1. ábra. A Look kezelő

A 8-1. ábrán van a Look kezelő. Ez a 10-es kezelő, és csak a MUTASD parancsszó hívja be. A következő két feladatot látja el:

- Részletesen leírja a helyiséget.
- Felsorolja a közelben található tárgyakat.

Igazában nem sokban különbözik attól, amit a Vezérlőleíró alárendelt része tesz, amikor belépünk egy helyiségbe. Look mindössze annyit tesz, hogy nem veszi figyelembe azt az információt, hogy jártunk-e már a helyiségben.

Két szubrutin játszik szerepet Look-ban. Az első Viewrm, amely egy helyiség hosszú vagy rövid leírását írja ki a C változóban tárolt értéktől függően. Ha C értéke nulla, akkor Viewrm a hosszú leírást írja ki. (Look nullát tesz C-be, és meghívja az 1160-as sort.)

Figyeljük azonban meg, hogy Viewrm végrehajtja a láthatóság eldöntéséhez szükséges ellenőrzéseket! Ha a játékos a föld színe alatt van, és nincs fáklyája, a Viewrm nem jeleníti meg a leírást, hanem figyelmezteti a játékost: „TÚL SÖTÉT VAN! BELEESHET EGY GÖDÖRBE!"

A másik szubrutin az 1140-es sorban található Listob. A Listob sorraveszi a tárgyak listáját, kikeresi azokat a tárgyakat, amelyek a helyiségben vannak, és ezeket leírja a játékos számára. Ebben az esetben sem jelenik meg a leírás, ha túl sötét van. Ilyenkor azonban nem jelenik meg figyelmeztető üzenet.

Nem haszontalan átismételni a Listob és Viewrm működését. Leírásuk megtalálható a Vezérlő működését részletező fejezetben.

Leltárt készítünk

Jó, ha az ember tudja, mije van. Amikor szembetalálja magát a dühös Kardhallal, nem biztos, hogy tudja, birtokában van-e a szekerce. Esetleg amikor kincsre bukkan, tudatják vele, hogy túl sok tárgy van már nála. Döntenie kell tehát, mit tartson meg és mit dobjon el.

```
NÉV:          INVEN
TIPUS:        KEZELŐ
FELADAT:      A hordott tátgyak
              felsorolása

340 B=18:GOSUB1100:FORJ=1TO16:IFOB(J,1)<>21
THENNEXTJ:GOTO104:ELSEA=4:B=J:GOSUB1040:REA
DB$,B$:PRINTB$:NEXTJ:GOTO104
```

8-2. ábra. Az Inven kezelő

A 8-as kezelő kényelmes lehetőséget biztosít arra, hogy megvizsgáljuk a kalandozó hátizsákját. A kezelő neve Inven, és a 8-2. ábrán látható.

Lényegében az Inven leltárt készít mindenről, amit a játékos éppen magával visz. A Listob-hoz nagyon hasonló módon, ez a kezelő is átnézi a tárgyak állapotmátrixát, hogy kikeresse a hátizsákban található dolgokat. Ezeket a tárgyakat röviden leírja.

Az Inven a 18-as bevezető üzenettel kezdődik, amely így szól: „A KÖVETKEZŐ DOLGOK VANNAK ÖNNÉL:". Ezután egy FOR-NEXT ciklust futtatunk le, és minden tárgy helyét összehasonlítjuk 21-gyel, a hátizsáknak megfelelő hellyel. Amelyik tárgy helye 21-gyel egyenlő, annak rövid nevét kikeresi és kiírja.

Minden tárgyhoz két leírás tartozik: egy legföljebb 64 karakteres hosszú és egy egy-két szavas rövid. Ezeket párosával tárolja a program, soronként egy párt a DATA utsítások között, a tárgyleírások blokkjában. Az Inven-nek minden kiírandó tárgy esetén meg kell találnia a megfelelő sort és a rövid nevet.

Az Inven az 1040-es sorban található Access segítségével találja ezt meg. Az Access az A változóban várja az adatblokk számát és a B változóban a blokkon belüli sorszámot. A tárgyakat leíró blokk a 4-es; ennek megfelelően állítja be A-t. Mivel a tárgy száma J-ben van, és a leírások ennek a számnak megfelelően soronként vannak tárolva, az Inven B-be tölti J-t. Amikor az Access lefut, az Inven elkezdi az adatok olvasását és azonnal megtalálja a tárgy rövid és hosszú leírását.

Az Inven-t a két leírás közül csak a második érdekli, a rövidebb. A „READ B$, B$" utasítás eredményeként a rövid név B$-ba kerül. A kezelő kiírja ezt a nevet, azután a ciklus folytatódik. Ez az eljárás addig ismétlődik, amíg az összes tárgyat meg nem vizsgáltuk, vajon a zsákban van-e.

Talán következetlenségnek tűnik, de figyeljük meg, hogy a játékos bármikor készíthet leltárt — teljes sötétben is. (Feltehetjük, hogy méretük és formájuk szerint képes azonosítani a tárgyakat!)

Sárkányok–hősök 10:0

Miközben a kalandozó egyik veszélytől a másikig halad, tudnia kell, hogy áll a küzdelem — vagy azért, hogy továbbhaladásra ösztönözze, vagy figyelmeztetésképpen, hogy ideje megkeresni a kifelé vezető utat, addig amíg életben van. A játékos úgy juthat ehhez az értékes segítséghez, hogy leírja a „PONTOZZ" parancsot, és máris látja, hányadán áll.

A *Kardhalak és kincsek* a következő, nagyon egyszerű pontozási rendszert használja:

- 5 pont jár minden felkeresett helyiségért,
- 10 pont jár a támaszpontra vitt minden egyes kincsért,
- 20 pont jár minden megölt passzív lényért,
- 25 pont jár minden megölt kitartó lényért (Kardhalért),
- 20 pont levonás jár a kalandozó halála esetén.

A 8-3. ábrán látható a program két részletének kódja. A 420-as sor a Score kezelő, aminek behívására akkor kerül sor, amikor a billentyűkön leírjuk: PONTOZZ. A Score mindössze annyit tesz, hogy meghívja a Points nevű

```
NÉV:              SCORE
TIPUS:            KEZELŐ
FELADAT:          A jelenlegi pontszám
                  kiírása

420 GOSUB1240:GOTO104

NÉV:              POINTS
TIPUS:            SZUBRUTIN
BEMENŐ ADAT:      nincs
EREDMÉNY:         Kiszámítja a jelenlegi
                  állást és kiírja

1240 B=30:GOSUB1100:A=CT(4):FORI=1TO20:IFRI
GHT$(STR$(RM(I)),1)="1"THENA=A+5
1242 NEXTI:FORI=1TO8:IFOB(I,1)=1THENA=A+10
1244 NEXTI:FORI=13TO16:IFOB(I,1)=0THENA=A+2
0
1246 NEXTI:A=A-CT(3)*20:PRINTA:PRINTCT(1);"
Lépés":RETURN
```

8-3. ábra. A Score kezelő és a hozzá kapcsolódó Points szubrutin

szubrutint, amely szintén a 8-3. ábrán látható. Ennek az elrendezésnek az az oka, hogy más kezelők is használhassák Points-ot, nevezetesen a most folyó játékot lezáró kezelő.

A Points azzal kezdi, hogy kiírja a 30-as számú bevezető üzenetet: „PONTSZÁMA A KÖVETKEZŐ:". Ezután CT(4) tartalma A-ba kerül. Emlékezzünk az előző fejezetekből, hogy CT(4) tartja számon, hány kitartó lényt (Kardhalat) öltünk meg; a CT(4) 25-tel nő, valahányszor elpusztítunk egy Kardhalat. Az A változó tekintélyes pozitív értékről indul, ha sok Kardhalat öltünk meg a játék során.

Ezután a Points ellenőrzi az összes helyiség állapotát, és 5 pontot ad a játékos által felkeresett minden egyes helyiségért. A helyiségek állapotvektorában, RM(x)-ben, az egész szám 1. számjegye egy, ha már jártunk a helyiségben, egyébként nulla. A Points két lépésben választja le az 1. számjegyet: az egész számot először karakterlánccá alakítja az STR$ függvénnyel, azután a RIGHT$ függvénnyel leválasztja a jobb szélső karaktert. A Points minden egyes helyiségnél elvégzi ezt az elemzést 1-től 20-ig. Valahányszor egyet talál, a pontszámot öttel növeli.

Ezután a kincseket veszi számba. Egy kincs csak akkor számít bele a játékos pontjaiba, ha azt biztonságba helyezte az 1-es helyiségben; a támaszponton. Az 1 és 8 közötti tárgyak kincsek, ezért Points ezeknek a különleges tételeknek a helyzetét ellenőrzi. Miközben egy ciklussal megvizsgálja a tárgyak állapotmátrixát, 10 jutalompontot ad minden kincsért, amelyik az 1-es helyiségben van.

Most a dicső harcos learatja jutalmát. A hőssel négy passzív lény áll szemben és tetszőleges számú Kardhal. A Kardhalakat a szubrutin elején számításba vettük: most a megölt passzív bestiákat értékeljük ki.

Amikor egy ilyen lényt megölünk, úgy távolítjuk el a színről, hogy a nem létező nullás helyiségbe száműzzük. Így tehát a Points ennek a helynek a vizsgálatával egyszerűen meg tudja állapítani, hogy hány passzív ellenséget öltünk meg. A Points lefuttat egy ciklust, hogy megvizsgálja a 13-as és a 16-os közti szokványos lényeket. Azok a lények, amelyek a 0-s helyiségben vanak, egyenként 20 pontot jelentenek a kalandozó számára.

Végül, a játékos 20 pontot elveszít minden olyan esetért, amikor szakadékba zuhant, a lángok között megégett vagy felfalták. A CT(3) változóban tartjuk számon ezeknek az eseményeknek a számát, tehát meg kell szorozni 20-szal és a pontszámból le kell vonni.

Most végre a Points kiírja az eredményt! A képernyőn két adat jelenik meg. Először az A változót írja ki, ez az éppen érvényes pontszám. Másodiknak az eddig megtett lépések számát. A Vezérlő a CT(1) változóban gyűjti a megtett lépések számát; a Points tehát ennek értékét írja ki.

Lényegében a játékos kalandozói rátermettségét két mércével értékeljük. Első a pontszám maga. Ez a pontszám elérheti a 260-at, ha a játékos vigyáz rá, hogy meg ne haljon, de lehet ennél jóval több is, ha a véletlen összehozza néhány

„ajándék" Kardhallal, amelyek megöléséért alkalmanként 25 pont jár. A második mérce a hatékonyság.

A mohó kalandjátékos először magas pontszám elérésére törekszik. Ha ezt már elérte, akkor újra játszik, most már a sebességet és a lehető legkevesebb lépést tűzve ki célul.

Amikor minden más csütörtököt mond

A játékosnak mindig módjában áll a játékot befejezni, akár azért, mert már unja, akár azért, mert a sikertelenség elvette a kedvét a folytatástól. Persze megteheti, hogy drasztikusan megnyomja a BREAK billentyűt. Ez a befejezés leggyorsabb módja, de a színvonal kedvéért a *Kardhalak és Kincsek* tartalmazza a BEFEJEZEM parancsot.

Ha a játékos azt írja le, hogy „BEFEJEZEM", ezzel a 8-4. ábrán látható Quit nevű 14-es számú kezelőt hívja be. A Quit három igen egyszerű dolgot tesz. Búcsúüzenetet ír ki, az üzenetek blokkjának 31-es üzenetét, ami így hangzik: „VÁRJUK EGY ÚJABB JÁTSZMÁRA!" Hogy a játékos akar-e ezután újra játszani, ez már az ő gondja!

Másodszor, a Quit kiírja a játékos végső pontszámát, és a játék során megtett lépések számát. Láttuk már, hogyan megy ez a Points szubrutin segítségével. A Quit egyszerűen végrehajt egy Gosub 1240 utasítást, ezzel a feladat el is van intézve.

```
NÉV:           QUIT
TIPUS:         KEZELŐ
FELADAT:       Befejezi a játékot

430 B=31:GOSUB1100:GOSUB1240:END
```

8-4. ábra. A Quit kezelő

Végül, a Quit egyszer és mindenkorra befejezi a programot. Az END lehetetlenné teszi, hogy a programot CONT paranccsal folytassuk, tehát ne adjuk ki a BEFEJEZEM parancsot, ha nem gondoljuk komolyan.

A játék kimentése mágnesszalagra

Előfordul, hogy abba kell hagyni a játékot, de nem akarjuk befejezni. A vacsora nem várhat, pusztán azért, mert a játékos eltévedt egy Útvesztőben, ahol még egy Kardhal is rátámadt. Még a kalandozónak is szüksége van néha egy kis alvásra. Annak érdekében, hogy a játékos többé-kevésbé „mindennapi módon" élhessen, célszerű lehetőséget biztosítani arra, hogy a játékot jelenlegi állapotában mágnesszalagra menthesse, hogy aztán később folytatni tudja.

A *Kardhalak és kincsek*-ben két kezelő segíti elő, hogy mágnesszalagos tárolással megszakíthassuk a kalandozást. Az első a mágnesszalagra írja a létfontosságú változók tartalmát, a másik visszatölti ezeket a szalagról. Az első funkciót a MENTSD KI, a másodikat a TÖLTSD BE kulcsszó aktivizálja.

Lássuk, mit kell okvetlenül mágnesszalagra menteni ahhoz, hogy megőrizzük a játék mostani állását. A Save kezelő a következő változók tartalmát menti ki:

- a helyiség számát, ahol a játékos tartózkodik,
- az eddig megtett lépések számát,
- a magával vitt tárgyak számát,
- eddig hányszor halt meg a játékos,
- az eddig megölt Kardhalak számát,
- a helyiségek állapotát,
- az összes tárgy állapotát.

A kezelő megírásakor, mint hamarosan látni fogjuk, egy elég nehezen megoldható akadály bukkant fel. A nehézség a mágnesszalagos adatállományokat kezelő PRINT≠–1 és INPUT≠–1 BASIC utasításokból származik. Ezt a két utasítást sokféle módon használhatjuk, egyik módszer hatékonyabb, a másik kevésbé.

Vegyük figyelembe, hogy a PRINT≠–1 utasítás legföljebb 255 byte-os csoportokban írja fel az adatokat a mágnesszalagra! Minden adatcsoportot egy körülbelül 5 másodpercnyi szinkronizáló jelsorozat előz meg. Ezért a mágnesszalagos műveletekre fordított idő legnagyobb részét a szinkronizáló jelek adják. Az előzőekből világosan következik, hogy a hatékony mágnesszalagos tárolás érdekében a szinkronizáló jelsorozatok számát a minimumra kell korlátozni.

Mivel minden adatcsoporthoz öt másodperces szinkronizáló jelsorozat tartozik, a cél az, hogy a szükséges adatokat minél kevesebb csoportba sűrítsük. Egy csoport hossza 1-től 255 byte-ig terjedhet. A szinkronizáló jelekre fordított idő csökkentése érdekében a Save kezelő a lehető legtöbb változó tartalmát sűríti egy csoportba.

* A magyar nyelvtanhoz igazodva a harmadik szót vesszük figyelembe (l. az *Előszó*-t).

A 8-5. ábrán két példát látunk arra, hogyan menthetjük több változó értékét mágnesszalagra. Az első példában a tíz elemű A(X) vektor tartalmát kell kimenteni. Egy FOR-NEXT ciklus segítségével ismételten végrehajtott PRINT≠–1 utasítással tároljuk a tíz elemet. A probléma a következő: valahányszor újra végrehajtjuk a PRINT≠–1 utasítást, egy új bevezető jelsorozat kerül a szalagra, és új adatcsoport kezdődik. Ennek az az eredménye, hogy az első módszer egy tíz külön csoportból álló hosszú adatállományt hoz létre, ahol minden adatcsoport öt másodpercig tart. Egy egyszerű vektor tárolása és betöltése külön-külön majdnem egy percig tart.

Most lássuk a 8-5. ábrán látható második példát! Ezúttal ahelyett, hogy ciklusban egyesével mentenénk a vektor elemeit, vesszővel elválasztva felsoroltuk mind a tíz elemet. Mind a tíz elemet egyetlen PRINT≠–1 utasítással tároljuk. Az eredmény, hogy a tíz vektorelemet egy csoportban tároljuk egyetlen bevezető jelsorozattal. (Tíz egész típusú változó egyenként körülbelül öt byte hosszú, ez összesen csak 50 byte.) Az előbbi egyperces tárolási vagy betöltési idő helyett, most csak öt–hat másodpercre van szükség. Micsoda különbség! És pont itt van a kelepce — a Save kezelőnek sok változót kell menteni. A mágnesszalag szempontjából a második módszer gyors és hatékony, de a BASIC kód és a tár szempontjából pocsékolás. Képzeljük csak el, hogy a helyiségek állapotvektorának mind a 20 elemét, az akadálylista tíz elemét, és így tovább, egyesével felsoroljuk! Ez jó néhány sort elfoglalna, nem is beszélve a tárban elfoglalt byte-okról.

Szokás szerint egyfajta kompromisszumot kell kötni, amely nem olyan gyors, mint lehetne, és nem is olyan helyigényes, mint amit a többletsebesség igényelne. A végleges változat a 8-6. ábrán látható; ez a lényeges változókat kilenc tömött csoportban menti ki és a mentés körülbelül 48 másodpercig tart. Ha ezt FOR-NEXT ciklusokkal vessző elválasztók nélkül oldanánk meg, ez 52 rosszul kihasznált csoportot és jóval több mint négy percet jelentene.

A Save kezelő először a 19-es üzenettel figyelmezteti a játékost: „ÁLLÍTSA FELVÉTELRE A MAGNETOFONT, MAJD ÜSSE LE A NEW LINE BILLENTYŰT". Az ezt követő INPUT A utasítás felfüggeszti a kezelő futását,

```
20 FORI=1TO10:PRINT#-1,A(I):NEXT
22 PRINT#-1,A(1),A(2),A(3),A(4),A(5),A(6),A
(7),A(8),A(9),A(10)
```

8-5. ábra. Két módszer indexes változó tartalmának kimentésére

```
NÉV:          SAVE
TIPUS:        KEZELO
FELADAT:      Mágnesszalagra menti a
              játék állását

500 B=19:GOSUB1100:INPUTA:FORI=0TO8:PRINT#-
1,OB(I,0),OB(I,1),OB(I+8,0),OB(I+8,1),RM(I+
8),RM(I+12),BK(I),BK(I+2),CT(I):NEXT:GOTO10
4
```

8-6. ábra. A Save kezelő

míg a NEW LINE billentyűt lenyomják, ezzel időt ad a kazetta berakására és a magnetofon beállítására.

Amint a NEW LINE billentyűt lenyomják, a kezelő végrehajt egy 0-tól 8-ig futó ciklust (a ciklus összesen kilencszer fut). Lássuk, hogyan menti ez a ciklus az összes tömböt! Bár bizonyos változókat valójában kétszer mentünk ki, mégis ez a többlet egyszerűbbé és hatékonyabbá teszi a ciklust.

Az első mentésre kerülő tömb a tárgyak állapotmátrixa. A PRINT≠–1 után álló két elem az OB(0,0)-tól OB(8,0)-ig és az OB(0,1)-től OB(8,1)-ig terjedő elemeket az egymást követő csoportokban menti ki.

Mi történik a tömb többi részével? A listán soron következő két elemben nyolcat adunk a ciklusváltozó értékéhez, a kimentendő elemek indexében. Ez így az OB(8,0)-tól OB(16,0)-ig és az OB(8,1)-től OB(16,1)-ig terjedő elemeket jelenti. Így ugyan kétszer került sor az OB(8,0) és OB(8,1) értékre, de a későbbiekből kiderül, hogy ezért a többletért bőségesen kárpótol az adatcsoportok jó kitöltöttsége.

Ezen túl az egyes adatcsoportokban menteni kell a helyiségek állapotvektorát. A lista egy eleme menti az RM(0) és RM(8) közötti elemeket. A következő elemnél a ciklusváltozó nyolccal megnövelt értéke az index, így ez az RM(8) és RM(16) közötti értékeket írja mágnesszalagra. A következő elemnél 12-t adunk a ciklusváltozóhoz, ez tehát az RM(12) és RM(20) közötti elemeket tárolja. Itt is megfigyelhetjük, hogy néhány elemet többször is tárolunk, de megéri. Az alapelv az, hogy egy FOR-NEXT ciklussal intézzünk el mindent, hogy a lehető legkevesebb ciklust használjunk. Az akadálylista következik ezután. A hozzá tartozó első elem a BK(0) és BK(8) közötti változókat menti. A következőnél kettőt adunk a ciklusváltozóhoz, így az a BK(2) és BK(10) közti elemeket tárolja. Ismét van fölöslegesen tárolt adat, cserében a dolog egyszerűségéért.

Az utolsó kimentésre kerülő vektor, a CT(X) tartalmazza a játékállást számontartó általános változókat. Ezek közül a legfontosabbak a CT(0) és CT(4) közöttiek, mindenesetre Save az összeset menti CT(8)-cal bezáróan.

A Save tehát mindössze kilenc adatcsoportba tömörítve kimenti a *Kardhalak és kincsek* összes fontos változóját. Ennek ellenére a csoportok nincsenek teljesen kihasználva, mivel mindegyik legföljebb 80 byte-ból áll — az elméletileg lehetséges értéknek ez csak egyharmada. A hatékonyság növelésének egyetlen módja az lenne, hogy csökkentjük a ciklus lefutásainak számát és növeljük az egyesével felsorolt tömbelemek számát. Írtunk egy olyan változatot is, amelyben a ciklus csak ötször futott — de kétszer ekkora helyet foglalt el a tárban!

Létezik a változókat mágnesszalagra mentő kezelőnek egy ellentett párja is, amelyik a változókat betölti a tárba, hogy a játékot folytatni lehessen. A Restore kezelő a 8-7. ábrán látható.

A Restore-t nem kell részletesen ismertetni, mivel legtöbb vonatkozásban a Save-vel azonos. Az egyetlen különbség, hogy a szalagra író PRINT≠–1 utasítás helyett a mágnesszalagról beolvasó INPUT≠–1 utasítás szerepel. A két kezelő között fennálló nagyfokú hasonlóság biztosítja, hogy minden változó helyesen lesz betöltve. A játék Save előtti állapota pontosan megegyezik az újrabetöltés utáni állapottal.

Pont a két kezelő nagyfokú hasonlósága miatt, jogosan vetődik fel a kérdés: nem lehetne-e valamilyen módon utasításokat megtakarítani? Nem lehetne valamit közössé tenni? Nem könnyű feladat, mivel mindkét kezelő legnagyobb része olyan szorosan kötődik a beolvasó, ill. a kiíró utasításhoz.

A program kidolgozása közben készült egy rövid programrészlet, amely egy POKE paranccsal INPUT≠–1 utasításra *változtatta* a PRINT≠–1 utasítást.

```
NÉV:                RESTORE
TIPUS:              KEZELŐ
FELADAT:            Mágnesszalagrol betölti a
                    játék állását

520 B=19:GOSUB1100:INPUTA:FORI=0TO8:INPUT#-
1,OB(I,0),OB(I,1),OB(I+8,0),OB(I+8,1),RM(I)
,RM(I+12),BK(I),BK(I+2),CT(I):NEXT:GOTO104
```

8-7. ábra. A Restore kezelő

```
NÉV:          LINERS
TIPUS:        KEZELŐ
FELADAT:      Egysoros választ ír ki
              bizonyos kulcsszavakra

360 CT(5)=N:GOSUB1000:B=CT(9)*10+CT(8):GOSU
B1100:GOTO104
```

8-8. ábra. A Liners kezelő

(Végül is mindkét utasítás egy-egy meghatározott egy byte-os kóddal van ábrázolva a tárban.) A kis programrészlet azonban bonyolult lett, és nem takarított meg annyi helyet a tárban, ami indokolta volna a bonyoldalmakat.

Frappáns megjegyzések

Hiszi-e az Olvasó vagy sem, mindössze egy olyan kezelő van hátra, amit nem vizsgáltunk meg! Ez gondoskodik egy olyan finomságról, ami nagyban emeli a játék színvonalát. Röviden, a kezelő találó válaszokat ad bizonyos beírt üzenetekre.

A Liners kezelő listája a 8-8. ábrán látható. Mindössze egy feladata van: meghatározott üzenetet kell kiírnia válaszul egy bizonyos parancsra.

Hogy a Liners-t használhassuk, a működését kiváltó kulcsszavakat tárolni kell a szótáblázatban. Emlékszünk rá, hogy a szótáblázatban szereplő igék az AZ szám 1. és 2. számjegyével mutatnak a megfelelő kezelőre! A liners működését kiváltó kulcsszavakban az 1. és 2. számjegyen „09" áll, mivel Liners a 9-es kezelő.

Néhány ige egy-egy különleges célra felhasználja a 3. és a 4. számjegyet. Pl. az irányt megadó igék az irányt adják itt át az Xmove kezelőnek. Egyszerűsíti a dolgokat, ha a Liners-t működtető kulcsszavak a 3. és 4. számjegyen adják át az üzenet sorszámát; azét az üzenetét, amely a beírt parancsra válaszul íródik ki.

Jelenleg a szótáblázatban mindössze két kulcsszó hívja a Liners-t. Az első a SEGÍTSÉG. Jó néhány nagyobb gépen futó kalandprogramban a SEGÍTSÉG parancs hatására terjedelmes leírást kapunk arról, hogyan is kell a játékot játszani. Sajnos, nekünk nem áll rendelkezésre akkora tár, hogy megengedhednénk ekkora fényűzést. Ehelyett bosszantsuk a játékost! A SEGÍTSÉG-hez tartozó 37-es üzenet így hangzik: „RIMÁNKODÁSA SÜKET FÜLEKRE TALÁL, SZÁNALMAS ALAK".

A Liners működését kiváltó másik parancs a VÁRJ. Néhány programban, ha a játékos leírja ezt a parancsot, várakozhat a helyzet megváltozására. Pl., ha a bűvös futóbab nő, a játékos várakozhat, és a bab a szeme láttára növekszik. A *Kardhalak és kincsek* ismét csak viccelődik. A 38-as üzenet kerül sorra, és ez így szól: „MÚLIK AZ IDŐ..."

A Liners kezelő működése egyszerű. Az N változó máris tartalmazza a beírt szó AZ számát, ezt a Vezérlő hathatós segítségének köszönhetjük. Liners a CT(5) változóba tölti N-t és meghívja az Analyz szubrutint, ami öt számjegyre bontja fel az AZ számot. Ezután Liners-nek le kell választania az AZ számba beágyazott üzenetsorszámot. A 3. és 4. számjegyet most a CT(8), ill. a CT(9) változó tárolja. A CT(9) * 10+CT(8) kifejezés ismét előállítja az üzenet sorszámát, amit a B változóban tárolunk. Végül meghívjuk Mesprt-t, amely kiírja a kívánt üzenetet.

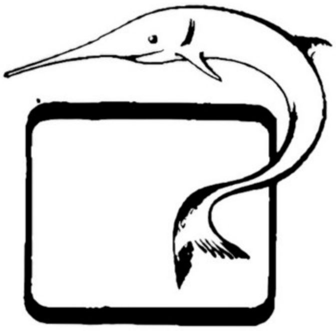
Még milyen egysoros válaszokkal egészíthetnénk ki a játékot? Néha, amikor egy játékos gyanítja, hogy a program felismer egy bűvös szót, kipróbálja a sajátját, pl. azt, hogy ABRAKADABRA. Ezt felvehetjük a szótáblázatba, hogy a következő választ váltsa ki: „EZ A BŰVÖS SZÓ MÁR A KÖNYÖKÖMÖN JÖN KI, ITT SEMMIRE SEM MEGY VELE!" Vagy pl. egy hídon álló játékos leírhatja az UGORJ parancsot. Erre a program pl. azt válaszolhatja: „MILYEN MAGASRA?" Szóval értjük a lényeget.

No fiúk, ez igazán egyszerű volt!

Bármilyen hihetetlen, az Olvasó befejezte a *Kardhalak és kincsek* kódjának átnézését. Láthatta, hogy ad a program a változóknak kezdőértéket.

Megvizsgálta a Vezérlőt és láthatta, hogyan írja le a helyiségeket és egyebeket. Végigkísérte a beírt kulcsszavakat a billentyűktől a szótáblázaton át a kezelőig, a háttérben statisztáló szubrutinokkal. Tanulmányozhattuk, hogyan kószál a kalandozó, kapukat nyitogatva, kincseket szedegetve, vadakat öldösve, miközben néha őt is megölik. Végül láthatta az Olvasó, hogyan bővítettük magát a programot különleges szolgáltatásokkal.

Mi van még hátra? Két végső feladat. Az egyik, hogy az eddigi részleteket összerakjuk egy teljes listává, táblázatokkal, jegyzetekkel kiegészítve stb., hogy végül betölthessük és futtassuk a *Kardhalak és kincsek*-et. Ezt megtesszük a következő fejezetben. A másik néhány javaslat, ami a program sebességét növeli vagy a hatékonyságát javítja. Ezt a rá következő fejezetre hagyjuk.



9. FEJEZET

A Kardhalak és kincsek listája

Végre, annyi tömény unalom, annyi gürcölés és locsogás után, magunk előtt látjuk a *Kardhalak és kincsek* teljes listáját. Mielőtt nekigyürkőznénk a begépelésnek, helyénvaló néhány megjegyzés.

Először is, az Olvasónak fel fog tűnni, hogy egyáltalán nincsenek REM utasítások a listában. (Remélem, a könyvben található megjegyzések fölöslegessé teszik, hogy a listán is legyenek megjegyzések.) Az ok nyilvánvaló: a megjegyzések helyet foglalnak a tárban, és mi épp ezzel próbálunk takarékoskodni.

KEZELŐ	SORSZÁM	KEZELŐ	SORSZÁM
XMOVE	200	REAO	400
IMOVE	220	SCORE	420
TAKE	240	SAY	460
DROP	260	QUIT	480
OPEN	280	SAVE	500
CLOSE	300	RESTORE	520
FIGHT	320	BOMB	540
INVEN	340	AARDVARK	560
LINERS	360	RESUR	580
LOOK	380		

9-1. ábra. A KARDHALAK ÉS KINCSEK kezelői

SZUBRUTIN	SORSZÁM	SZUBRUTIN	SORSZÁM
ANALYZ	1000	LISTOB	1140
SYNTHE	1020	VIEWRM	1160
ACCESS	1040	DARKCK	1180
GETCOM	1060	CKOBS	1200
IDWORD	1080	REVOBS	1220
MESPRT	1100	POINTS	1240
TRAVEC	1120		

9-2. ábrs. A KARDHALAK ÉS KINCSEK szubrutinjai

Az már inkább zavaró, hogy nincsenek a szövegben szóközök, kivéve azokat a sorokat, amelyekben képernyőre írandó szövegek vannak. A dologban ismét az a ráció, hogy helyet takarítsunk meg, mivel a BASIC-nek igazából nincs is szüksége mindenütt a szóközökre. Biztos, hogy ez megnehezíti a szöveg beírását! Minden egybemosódik, a változóneveket nehéz a BASIC kulcsszavaktól elkülöníteni, és így tovább. A szavak és a kifejezések a sorok határán megtörhetnek.

Nagyon ügyeljünk a beírásnál! Ha a billentyűzet hajlamos az ismétlésre, fél szemmel nézzük a képernyőt, mert az ilyen jellegű hibákat nehezebb megtalálni a tömören írt programsorokban. Szinte azt mondhatom, hogy beírás előtt ajánlatos a sorokat elolvasni, a sort megfejteni, hogy felkészüljünk azokra a kifejezésekre, amelyeket könnyen elírhatunk.

Végül esetleg szükségét érezheti az Olvasó, hogy egyes sorokba szóközöket iktasson be a jobb érthetőség kedvéért; bátran tegye meg. De ha megteszi, azt javasoljuk, hogy ne tegyen szóközöket a szótáblázatba! Hogy miért? Mert a következő fejezetben megírunk egy gépi kódú szubrutint, amely igen gyorsan végigpásztázza a szótáblázatát. A szubrutin feltételezi, hogy nincsenek szóközök a táblázatban, és ha már most kihagyjuk őket, akkor később nem kell őket kiszedni.

A listák előtt — a könnyebb tájékozódás érdekében — két táblázat található. Az egyik felsorolja a kezelőket, a másik a szubrutinokat, növekvő sorszám szerint rendezve. Ez részben pótolhatja a REM utasítások hiányát, amennyiben megkönnyíti a kód egyes részeinek megtalálását.

```
2 CLS:PRINTCHR$(23):PRINTé466,"Köszöntjük a
":PRINTé522,"KARDHALAK éS KINCSEK":PRINTé59
7,"játékban!"
4 CLEAR500:DEFINTA-Z:DIMTX$(4),DA(5),RM(20)
,OB(16,1),BK(10),CT(12):FORI=1TO20:READRM(I
):NEXT:FORI=1TO16:READOB(I,1),OB(I,0):NEXT:
FORI=1TO10:READBK(I):NEXT
6 P=16548:N=1:FORI=5000TO9000STEP1000
8 IFI=PEEK(P+2)+PEEK(P+3)*256THENDA(N)=P:N=
N+1:NEXTI:GOTO10:ELSEP=PEEK(P)+PEEK(P+1)*25
6:IFP=0THENCLS:PRINT"ERROR":END:ELSE8
10 CT(0)=1:CT(12)=RND(10)+10:CLS
100 CT(5)=RM(CT(0)):GOSUB1000:C=CT(6):GOSUB
1160:GOSUB1180:IFB=0ANDC=0THENCT(6)=1:GOSUB
1020:RM(CT(0))=CT(5):ELSEIFB=1THENN=RND(100
):IFN<20THENB=5:GOSUB1100:GOTO580
102 GOSUB1140
104 GOTO112
105 INPUTA$
106 GOSUB1060:A$=TX$(2):GOSUB1080
108 CT(5)=N:GOSUB1000:IFCT(10)=0ORN=0THENB=
7:GOSUB1100:GOTO104
110 ONCT(6)+CT(7)*10GOTO200,220,240,260,280
,300,320,340,360,380,400,420,460,480,500,52
0,540,560,580,600,620,640,660,680,700
112 IFOB(0,0)=0ANDCT(0)>2THENCT(12)=CT(12)-
1:IFCT(12)<=0THENCT(12)=RND(10)+10:OB(0,1)=
CT(0):OB(0,0)=1:GOTO116:ELSE105
114 IFCT(0)<3THENOB(0,0)=0:GOTO105:ELSEOB(0
,1)=CT(0)
116 B=42:GOSUB1100:B=RND(100):IFB>75THEN105
ELSEB=43:GOSUB1100:B=RND(100):IFB>60THENB=4
4:GOSUB1100:GOTO580:ELSE105
200 D=CT(8)+CT(9)*10-1:FORK=1TO10:CT(5)=BK(
K):GOSUB1000:IFD<>CT(8)ORCT(0)<>CT(6)+CT(7)
*10THENNEXTK:GOTO202:ELSEIFBK(K)<0THEN202EL
SEB=CT(9):GOTO206
202 DEFSNG=D+1:GOSUB1120:IFA=22THENB=4:GOTO
```

```
204:ELSEIFA=23THENB=5:GOTO204:ELSEIFA=0THEN
B=6:GOTO206:ELSECT(0)=A:CT(1)=CT(1)+1:GOTO1
00
204 GOSUB1100:GOTO580
206 GOSUB1100:GOTO104
220 IFTX$(3)=""THEND=11:GOSUB1120:N=A*100+1
0101:GOTO108:ELSEA$=TX$(3):GOTO106
240 IFCT(2)>=5THENB=36:GOSUB1100:GOTO104:EL
SEA$=TX$(3):GOSUB1080:IFN>9999THENB=7:GOTO2
42:ELSEIFN>12ANDN<17ORN=18THENB=40:GOTO242
241 IFN=17THENB=8:GOTO242:ELSEIFOB(N,1)=21T
HENB=9:GOTO242:ELSEIFOB(N,1)<>CT(0)ORN=0THE
NB=12:GOTO242:ELSEOB(N,1)=21:B=11:CT(2)=CT(
2)+1
242 GOSUB1100:GOTO104
260 A$=TX$(3):GOSUB1080:IFN>9999THENB=7:GOS
UB262:ELSEIFOB(N,1)<>21THENB=10:GOTO262:ELS
EIFN=12THEN540:ELSEOB(N,1)=CT(0):B=11:CT(2)
=CT(2)-1
262 GOSUB1100:GOTO104
280 IFTX$(3)=""THENB=7:GOTO284:ELSEA$=TX$(3
):GOSUB1080:CT(5)=N:GOSUB1000:A=CT(8):GOSUB
1200:IFA=0THENB=12:GOTO284:ELSEIFBK(A)<0THE
NB=13:GOTO284:ELSEIFOB(11,1)<>21THENB=16:GO
TO284:ELSEGOSUB1220:B=12+CT(9)
284 GOSUB1100:GOTO104
300 IFTX$(3)=""THENB=7:GOTO304:ELSEA$=TX$(3
):GOSUB1080:CT(5)=N:GOSUB1000:A=CT(8):GOSUB
1200:IFA=0THENB=12:GOTO304:ELSEIFBK(A)>0THE
NB=13:GOTO304:ELSEGOSUB1220:B=17
304 GOSUB1100:GOTO104
320 IFOB(0,0)=1THEN322ELSEFORK=13TO16:IFOB(
K,1)<>CT(0)THENNEXTK:B=41:GOSUB1100:GOTO104

322 IFOB(10,1)<>21THENB=23:GOTO326:ELSEIFK=
15ORK=16THENB=24:GOTO324:ELSEX=RND(100):IFO
B(0,0)=1THEN328ELSEIFX>70THENB=26:GOTO324:E
LSEOB(K,1)=0:A=1:GOSUB1200:GOSUB1220:B=25:G
OTO326
324 GOSUB1100DEFSNGB=27:GOSUB1100:X=RND(100
```

```
):IFX<40THENB=29:GOSUB1100:GOTO580:ELSEB=28

326 GOSUB1100:GOTO105
328 IFX>70THENB=26:GOSUB1100:GOTO112:ELSEOB
(0,0)=0:OB(0,1)=0:B=25:CT(4)=CT(4)+25:GOTO3
26
340 B=18:GOSUB1100:FORJ=1TO16:IFOB(J,1)<>21
THENNEXTJ:GOTO104:ELSEA=4:B=J:GOSUB1040:REA
DB$,B$:PRINTB$:NEXTJ:GOTO104
360 CT(5)=N:GOSUB1000:B=CT(9)*10+CT(8):GOSU
B1100:GOTO104
380 C=0:GOSUB1160:GOSUB1140:GOTO104
400 IFCT(0)<>6THENB=32:GOTO402:ELSEB=33
402 GOSUB1100:GOTO104
420 GOSUB1240:GOTO104
460 IFLEFT$(TX$(3),5)<>"AARDV"THENB=34:GOSU
B1100:GOTO104:ELSE560
480 B=31:GOSUB1100:GOSUB1240:END
500 B=19:GOSUB1100:INPUTA:FORI=0TO8:PRINT#-
1,OB(I,0),OB(I,1),OB(I+8,0),OB(I+8,1),RM(I+
8),RM(I+12),BK(I),BK(I+2),CT(I):NEXT:GOTO10
4
520 B=19:GOSUB1100:INPUTA:FORI=0TO8:INPUT#-
1,OB(I,0),OB(I,1),OB(I+8,0),OB(I+8,1),RM(I)
,RM(I+12),BK(I),BK(I+2),CT(I):NEXT:GOTO104
540 IFOB(12,1)<>21THENB=20:GOTO544:ELSEOB(1
2,1)=CT(0):CT(2)=CT(2)-1:FORK=15TO16:IFOB(K
,1)<>CT(0)THENNEXTK:B=21:GOTO544:ELSEOB(K,1
)=0:A=1:GOSUB1200:GOSUB1220:B=22
544 GOSUB1100:GOTO104
560 IFCT(0)=6THENCT(0)=1ELSEIFCT(0)=1THENCT
(0)=6ELSEB=34:GOSUB1100
562 GOTO100
580 CT(3)=CT(3)+1:B=35:GOSUB1100:OB(9,1)=2:
FORI=1TO12:IFOB(I,1)=21THENOB(I,1)=CT(0):NE
XT:ELSENEXT
592 CT(0)=1:CT(2)=0:GOTO100
1000 FORZ=6TO10:CT(Z)=0:NEXTZ:B$=MID$(STR$(
CT(5)),2):FORZ=1TOLEN(B$):CT(6+LEN(B$)-Z)=V
AL(MID$(B$,Z,1)):NEXTZ:IFCT(5)<0THENCT(11)=
```

```
-1:RETURN:ELSECT(11)=1:RETURN
1020 CT(5)=CT(10)*10000+CT(9)*1000+CT(8)*10
0+CT(7)*10+CT(6):CT(5)=CT(5)*CT(11):RETURN
1040 P=DA(A):IFB=1THEN1042ELSEFORZ=1TOB-1:P
=PEEK(P)+PEEK(P+1)*256:NEXTZ
1042 P=P-1:POKE16640,FIX(P/256):POKE16639,P
-FIX(P/256)*256:RETURN
1060 FORI=1TOLEN(A$):IFMID$(A$,I,1)<>" "THE
NNEXTI:TX$(3)="":TX$(2)=A$:RETURN:ELSETX$(2
)=LEFT$(A$,I-1):FORI=LEN(A$)TO1STEP-1:IFMID
$(A$,I,1)<>" "THENNEXTIELSETX$(3)=MID$(A$,I
+1):RETURN
1080 IFLEN(A$)>5THENA$=LEFT$(A$,5)
1082 A=2:B=1:GOSUB1040
1084 READB$,N:IFB$="."ORB$=A$THENRETURNELSE
1084
1100 A=3:GOSUB1040:READA$:PRINTA$:RETURN
1120 B=CT(0):A=1:GOSUB1040:FORY=1TDD:READA:
NEXTY:RETURN
1140 GOSUB1180:IFB=1THENRETURN:ELSEA=4:FORB
=1TO16:IFCT(9)<>OB(B,1)THENNEXTB:RETURN:ELS
EGOSUB1040:READTX$(4):PRINTTX$(4):NEXTB:RET
URN
1160 GOSUB1180:IFB=1THENB=39:GOSUB1100:RETU
RN:ELSEA=5:B=CT(0):GOSUB1040:READTX$(0),TX$
(1):IFC=0THENPRINTTX$(0):RETURN:ELSEPRINTTX
$(1):RETURN
1180 IFOB(9,1)<>21ANDCT(0)<>1ANDCT(0)<>2THE
NB=1ELSEB=0
1182 RETURN
1200 FORQ=1TO10:CT(5)=BK(Q):GOSUB1000:IFCT(
6)+CT(7)*10<>CT(0)ORCT(9)<>ATHENNEXTQ:A=0:E
LSEA=Q
1202 RETURN
1220 BK(A)=-BK(A):CT(5)=BK(A):GOSUB1000:IFC
T(10)=1RETURNELSEBK(A-1+CT(10))=-BK(A-1+CT(
10)):RETURN
1240 B=30:GOSUB1100:A=CT(4):FORI=1TO20:IFRI
GHT$(STR$(RM(I)),1)="1"THENA=A+5
1242 NEXTI:FORI=1TO8:IFOB(I,1)=1THENA=A+10
```

```
1244 NEXTI:FORI=13TO16:IFO(I,1)=0THENA=A+2
0
1246 NEXTI:A=A-CT(3)*20:PRINTA:PRINTCT(1);"
Lépés":RETURN
2000 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
3000 DATA4,0,7,0,20,0,11,0,5,0,19,0,7,0,5,0
,2,0,3,0,10,0,12,0,4,0,18,0,14,0,6,0
4000 DATA22902,2808,23306,3712,23404,3011,1
1104,11118,11714,11306,0
5000 DATA1,2,2,1,1,1,1,1,0,3,9
5002 DATA2,3,2,2,2,1,1,2,0,8,9
5004 DATA0,0,4,10,0,0,0,0,1,0,8
5006 DATA0,5,0,0,11,0,3,0,0,0,4
5008 DATA0,0,0,0,0,4,0,0,0,0,5
5010 DATA0,0,0,12,0,0,0,0,0,23,3
5012 DATA0,0,0,14,0,0,0,0,0,0,3
5014 DATA0,0,0,0,14,0,0,0,2,0,8
5016 DATA9,0,36,25,9,0,0,9,0,0,7
5018 DATA23,23,23,16,17,16,17,3,0,17,4
5020 DATA4,0,0,0,0,0,0,0,0,0,0
5022 DATA0,0,13,0,18,0,0,6,0,0,7
5024 DATA0,0,0,0,0,0,12,0,0,0,6
5026 DATA8,0,0,0,19,0,0,7,0,19,4
5028 DATA15,0,15,0,15,16,9,0,0,0,0
5030 DATA15,16,16,0,16,0,10,9,0,0,1
5032 DATA18,18,18,18,18,18,18,18,18,18,0
5034 DATA12,19,0,0,0,0,0,0,0,0,0
5036 DATA14,0,0,0,0,18,0,0,14,20,9
5038 DATA22,22,22,22,22,22,22,22,19,22,8
6000 DATAÉKKÖV,1,KORON,1,ARANY,2,KOCKA,2,GY
ÉMA,3,BOGAR,3,EZÜST,4,ÖVET,4,PLATI,5,GYÜRÜ,
5,ONIXO,6,ÉRMET,7,HOMOK,8
6001 DATAFAKLY,9,SZEKE,10,KULCS,11,GRANA,12
,MANOT,13,MANOV,13,GYIKO,14,GYIKK,14,POKOT,
15,POKKA,15,NEVEN,1,BORZA,16,KARDH,18
6002 DATASARAT,17,IRDAS,17,SZEKR,17,HOLTT,1
7,COLAT,17,AUTOM,17,POKHA,17,ALLVA,17,KOPOR
,17,KAPUT,317,RÁCSO,217,F,10901,L,11001,É,1
0101,ÉK,10201,K,10301,DK,10401,D,10501,DNY,
```

```
10601,NY,10701,éNY,10801
6003 DATAéSZAK,10101,DéL,10501,KELET,10301,
NYUGA,10701,FEL,10901,LE,11001,PONTO,10012,
BEFEJ,10014,OLD,10007,KÜZDJ,10007,PUSZT,100
07,ROBBA,10017,BOMBA,10017,DéLNE,10501,DéLR
E,10501
6004 DATAVARJ,13809,SEGIT,13709,OLVAS,10011
,MONDD,10013,ZARD,10006,NYISD,10005,TARD,10
005,CSUKD,10006,ZARD,10006,MUTAS,10010,LELT
A,10008
6005 DATAVEDD,10003,EJTSD,10004,DOBD,10004,
LOPD,10003,BE,10002,KI,10002,MENJ,10002,LéP
J,10002,MENTS,10015,TOLTS,10016,AARDV,10018
,•,0
7000 DATA"A lény nem engedi át!"
7001 DATA"A rács be vàn csukva és le van la
katolva!"
7002 DATA"A kapu szorosan be van zárva és l
e van lakatolva."
7003 DATA"A lángok martalékává vált!"
7004 DATA"Halálra zuzta magát..."
7005 DATA"Arra nem mehet"
7006 DATA"Mit mond?"
7007 DATA"Sikertelenül probálkozik ... nem
mozdithato!"
7008 DATA"Már önnél van!"
7009 DATA"Nincs önnél!"
7010 DATA"Rendben."
7011 DATA"Nem látok itt semmi hasonlot."
7012 DATA"Nem szükséges."
7013 DATA"A rács csikorogva kiesik a helyéb
öl."
7014 DATA"A kapu szélesre tárul."
7015 DATA"Nincs kulcsa!"
7016 DATA"Becsapodik és a zár bekattan."
7017 DATA"A következö dolgok vannak Önnél:"

7018 DATA"Allitsa felvételre a magnetfofont
 majd üsse le a <NEW LINE> billentyüt."
7019 DATA"Nincs is bombája!"
```

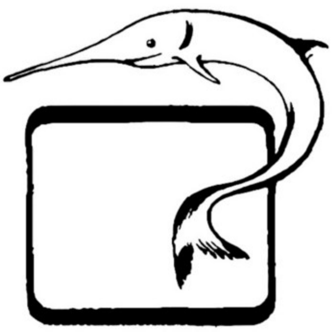
```
7020 DATA"A gránát a földre hull, semmi sem
 történik."
7021 DATA"A gránát hangtalanul, természetfö
lötti kék lobbanással
felrobban ... a lénynek vége!"
7022 DATA"Milyen fegyverrel?"
7023 DATA"Szekerce csapásai eröteljesek ...
 de hatástalanok!"
7024 DATA"A büvös szekerce lesujt! A lény e
ltünik
egy büzös füstfelhöben!"
7025 DATA"Elhibázta! PFUJ!"
7026 DATA"A förtelmes szörny a torkának ugr
ott!"
7027 DATA"Valahogy sikerült elháritani!"
7028 DATA"Végzett Önnel!!"
7029 DATA"Pontszáma a következö:"
7030 DATA"Várjuk egy ujabb játszmára!"
7031 DATA"Nincs itt semmi olvasnivalo ... o
h de unalmas!"
7032 DATA"A veszély
itt nem csekély
de mondd AARDVARK
ez a segély!"
7033 DATA"Semmi sem történik."
7034 DATA"Nos, dicsö kalandozo! Igazán nagy
 bajban van!
Szerencsére modunkban áll, hogy ujraélesszü
k Önt!
                        ... PUFF!!..."
7035 DATA"Karjai tele vannak...nem bir el t
öbbet."
7036 DATA"Rimánkodása süket fülekre talál,
szánalmas alak."
7037 DATA"Mulik az idö..."
7038 DATA"Tul sötét van! Beleeshet egy gödö
rbe!"
7039 DATA"Csuda öngyilkos hajlamokat mutat,
 apuskám!"
7040 DATA"Csak semmi pánik! Nincs itt semmi-
```

```
 veszély!"
7041 DATA"Egy dühös kardhal van a közelben!
"
7042 DATA"Elöre ront egy fekete görbe kardh
al!"
7043 DATA"A kardhal miszlikbe apritja."
8000 DATA"Itt egy ékköves korona!","Ékköves
 korona"
8001 DATA"Itt egy arany kocka!","Arany kock
a"
8002 DATA"Itt egy bogár formáju gyémánt!","
Gyémánt bogár"
8003 DATA"Itt egy pompás ezüst öv!","Ezüst
öv"
8004 DATA"Itt egy tiszta platina gyürü!","P
latina gyürü"
8005 DATA"Itt egy darab csiszolt onix!","On
ix"
8006 DATA"Itt egy milliokat érö érme!","Érm
e"
8007 DATA"Itt egy antik homokora!","Homokor
a"
8008 DATA"Itt egy égö fáklya!",Fáklya
8009 DATA"Itt egy hatalmas büvös szekerce!"
,Szekerce
8010 DATA"Itt egy oriási kulcs",Kulcs
8011 DATA"Itt egy büvös gránát.",Gránát
8012 DATA"Egy oriási ájtatos meno laplul a
 közelben, ugrásra készen!"
8013 DATA"Egy hatalmas gyik toporzékol a kö
zelben. Önt figyeli!"
8014 DATA"Egy hatalmas fehér pok tornyosul
Ön fölé, szája rángatozik!"
8015 DATA"A nevenincs borzalom elöbujik egy
 gödörbol. Gyilkos csápjaival
elzárja a visszavonulás utját!!"
9000 DATA"Egy nagy gödör alján áll. Lábainá
l egy szuk nyilás van,
pont átfér rajta.","Gödör alja"
9001 DATA"A törpék odon kastélyának romjai
```

```
láthatok itt. A közelben egy
rács, azon tul a sötétség...",Romok
9002 DATA"Jol látható, hogy valaha itt egy
fegyvertár volt. Ma már a
tartok üresek. A plafonon egy lyuk van, kel
et felé egy boltiv
vezet, a délkeleti falon egy csipkés lyuk v
an.",Fegyvertár
9003 DATA"Világosan kivehetök a törpék és a
z emberformáju szörnyek közt
lezajlott csata nyomai...A jelek szerint a
törpéket legyüzték.
Mindenütt holttestek. Egy csipkés lyuk van
nyugatra, egy csarnokészakkeletre, dél felé
 pedig egy kapu.","Vesztes csata"
9004 DATA"A falakat koporsok szegélyezik...
ugy látszik, ez a törpék temetöje. Délnyuga
t felé egy kapu vezet.",Kripta
9005 DATA"Ez a kis terem a varázslat levegö
jét árasztja magábol.A jos
üzenetet hagyott a falon. Délkelet felé egy
 kapu vezet,
közelében egy nagy gödör van.",Josda
9006 DATA"Végre-valahára! A kincstár! Micso
da szégyen, hogy annyi
mindent elhordtak már az eredeti kincsböl!
Délkelet felé egy
kapu vezet ki.",Kincstár
9007 DATA"Valaha ez volt a földalatti törpe
-királyság fö örhelye.
A mennyezeten van a bejárati rács, dél felé
 egy kap u vezet.",Örhely
9008 DATA"Eltévedt egy utvesztöben!","Eltév
edt egy utvesztöben!"
9009 DATA"Egy északkelet, délnyugat irányu
keskeny sziklapárkányon halad.
Nyugatra, lenn a mélyben egy gyors folyo, k
eletre egy feneketlenszakadék!","Keskeny sz
iklapárkány"
9010 DATA"Egy szük börtöncellában van. A rá
```

```
cson keresztül egy szép iroda láthato...elé
rhetetlen. északra egy kapu lathato.",Cella

9011 DATA"Ez itt egy iroda. Az iroasztalok
és az irattartok üresek,
kifosztottak. A falba vágott rácsos ablakon
 át egy börtöncella
látszik. Két kapu van: egyik északnyugatra
a másik keletre.
A déli falon egy sziklás lyuk van.",Iroda
9012 DATA"Ez az ebédlö. Még Cola-automata i
s van...sajnos üres. Egy kapu
vezet nyugatra.",Ebédlö
9013 DATA"Micsoda hátborzongato hely! Minde
nütt pokhálok! Egy kapu vezet
északra, egy csarnok északnyugatra, a padlo
n is van egy lyuk.","Pokhálos helyiség"
9014 DATA"Eltévedt egy utvesztöben!","Eltév
edt egy utvesztöben!"
9015 DATA"Eltévedt egy utvesztöben!","Eltév
edt egy utvesztöben!"
9016 DATA"Egy rohano, hideg folyoban kapálo
zik! Ugy tünik bármit is tesz, semmi sem tu
dja megakadályozni, hogy a közeli sziklás b
arlang
bejáratához sodrodjon!","Rohano folyo"
9017 DATA"Egy sötét nyálkás barlang homokjá
ban fekszik, egy folyo mellett.A falakat vi
sszataszito sár boritja. Egy lyuk tátong az
 északi
sziklákon, északkeleti irányban egy ösvény
láthato.","Nyálkás barlang"
9018 DATA"Izzadság patakzik az arcán, mert
egy gözölgö barlangban van. A
padlon levo lyukbol füst száll föl, a menny
ezeten egy másik lyuklátszik karnyujtásnyir
a. Láthato még egy délnyugatra vezetö
ösvény is.","Gözölgö barlang"
9019 DATA"Egy tüzes csucson lapul, a csucso
t lángok veszik körül! Feje
fölött egy fél méterrel van a mennyezet, ra
jta egy lyuk.
Kibirhatatlan a höség!","Tüzes csucs"
```



10. FEJEZET

Tovább tökéletesítjük a programot

A számítógép-programozó híres örökös elégedetlenségéről. Nem éri be azzal, hogy a játékost órákra leköti a valóság pontos másával. Ez mind smafu! A programozó számára az jelenti az igazi kihívást, hogy a programot minél jobbra, gyorsabbra, hatékonyabbra, elegánsabbra készítse el.

Bolondság lenne azt állítani, hogy a *Kardhalak és kincsek* jobb és hatékonyabb már nem is lehetne. Valójában egy csomó fogással még többet tudunk kihozni a programból. Ebben a záró fejezetben a BASIC-et a legvégső lehetőségekig kihasználjuk, és meglátjuk, milyen bonyolult dolgokat valósíthatunk meg nagy sebesség mellett kis tárigénnyel.

Természetesen figyelmeztetnem kell az Olvasót, hogy mindez hová vezet; a BASIC jó szolgálatot tett az előző fejezetekben, de a szívem legmélyén tudatában vagyok, hogy a *Kardhalak és kincsek*-et ki kellene bővíteni és gépi kódban újra megírni. A fejezet vége felé néhány megjegyzéssel útbaigazítást adunk, hogyan lehet ezt a feladatot megkísérelni.

Helyénvaló egy bevezető megjegyzés. Ebben a fejezetben több javítási lehetőséget és módosítást ismertetek a hozzájuk tartozó BASIC kóddal együtt. Ezek nem mindegyike fér össze az eddigiekkel oly módon, hogy az új sorokat egyszerűen hozzáírjuk az előzőekhez. Mielőtt megpróbálnánk egyszerre mindent megvalósítani, tekintsük át, hogy mely változók módosulnak, és a változás a kódnak mely részeit befolyásolja. Egy apró változás könnyen továbbgyűrűzhet, és kalandprogramunk máris a szintaktikai hibák tengerén hánykódik!

Egy gyorsabb módszer a szavak kikeresésére

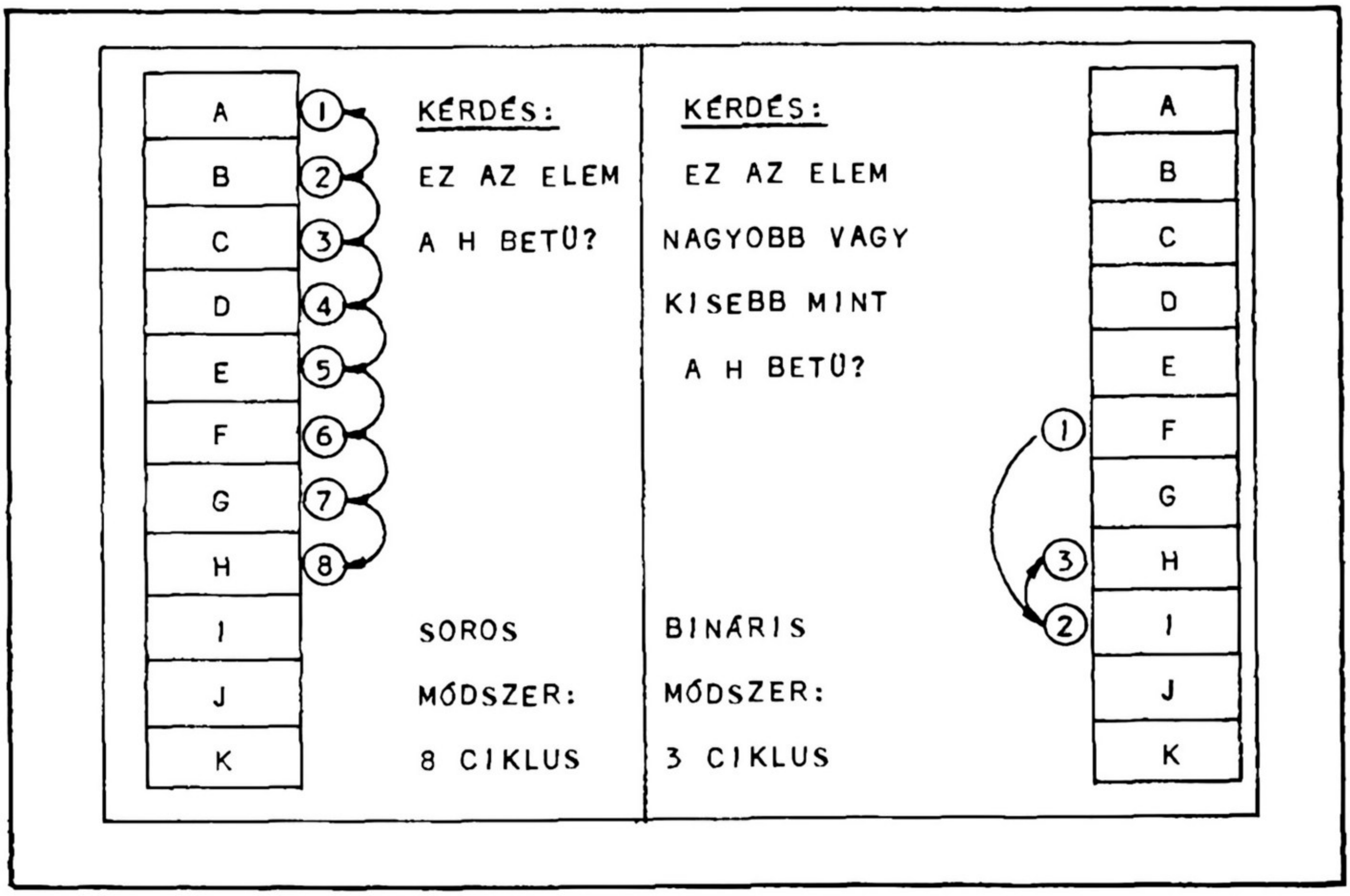
A 3. fejezetben megvizsgáltuk a Vezérlőparancs alárendelt részét. A kódnak ez a része először két dolgot tesz, ha beírnak egy egy-, kétszavas parancsot. A beírt szöveget először külön szavakra bontja (ha szükséges). Másodszor, veszi az első szót és megpróbálja azonosítani.

A szótáblázatként ismert adatblokk a bejövő üzenetek értelmezésének a kulcsa. Ebben a táblázatban fel van sorolva minden szó, amit a programnak fel kell ismernie, és mindegyikhez kapcsolódik egy ún. AZ szám, amely a szó értelmének meghatározásában segít. Ha egy szó nem található meg a szótáblázatban, akkor a program tudatlanságában egy ilyesfajta üzenettel válaszol: „MIT MOND?"

Az Idword nevű szubrutin feladata az, hogy a szavak táblázatát kezelje. Ha adott egy szó az A$ változóban, akkor az Idword átnézi a teljes táblázatot, és megpróbál olyan szót keresni, amely megegyezik A$ tartalmával. Ha talál ilyet, akkor az N változóba teszi a megfelelő AZ számot. Ha nem talál, akkor N-be nullát tölt.

Hogyan lehet a leggyorsabban átnézni egy táblázatot? Az egyszerű *soros keresés* az egyik módszer. A táblázat legelejétől kezdve minden szót megvizsgálunk, míg csak egyezést nem találunk. Ha nem fordul elő egyezés, akkor végigolvasni az egész táblázatot merő időpocsékolás.

Egy másik módszer a *bináris keresés*. Ennél a módszernél a táblázat szavait ábécésorrendbe szedjük. Először a középső elemet vizsgáljuk meg, egyezik-e? Attól függően, hogy a keresett szó ábécésorrendben megelőzi-e vagy követi-e



10-1. ábra. A soros és a bináris keresés összehasonlítása
(Figyeljük meg, hogy a bináris keresés megkeresi a vizsgált táblázatrész közelítő felezőpontját, ha kell fölfelé kerekít!)

a megvizsgált elemet, a táblázat egyik felét figyelmen kívül hagyhatjuk. A megmaradó fél középső elemét ismét megvizsgáljuk, és így tovább, amíg vagy előfordul egyezés, vagy nem. A bináris keresési módszer igazán gyors. (A 10-1. ábrán látható a két keresési módszer összehasonlítása.)

Mármost ezután a hatalmas előkészület után, ebben a fejezetben nem közlöm a szótáblázat bináris kereséssel vizsgáló szubrutinkódját. Hogy miért nem? Az elsődleges ok az, hogy a táblázat elemei DATA utasítások sorozatában vannak elhelyezve. Ezért még úgy is nehéz a táblázat közepén álló elemet, majd a táblázat felének közepén álló elemet megtalálni, hogy az adatok elérésére azt az ügyes módszert használjuk, hogy közvetlenül a BASIC adatmutatóba POKE-olunk. A READ utasítás természeténél fogva soros. Túl bonyolult kódot eredményezne, ha az adatmutató piszkálásával akarnánk a bináris keresést megvalósítani.

Akkor mit javasolhatok? Rendszerint egy jó kompromisszum több a semminél. Nézzük a 10-2. ábrát! Ott látható az Idword szubrutin javított változatának kódja. Az aláhúzott utasítások újak, a többi változatlan. Ezektől a bővítésektől eltekintve, még egy változtatásra van szükség: a táblázat szavait ábécésorrendbe kell rendezni. Ily módon az új Idword meg tudja állapítani, hogy nem jutott-e túl messzie a kívánt szó keresése közben, és mikor hagyhatja abba.

Az 1084-es sor a kritikus pont. Figyeljük meg, hogy a keresés a táblázat legelején kezdődik, mint egy szabályos soros keresésnél! A keresés minden egyes lépésében a táblázat egyik szavát beolvassuk B$-ba, a hozzá tartozó AZ számot pedig N-be. A táblázat legutolsó szava egy pont, a hozzá tartozó AZ szám nulla. Ezért minden egyes lépésben az Idword két feltételt ellenőriz. Ha B$ és A$ megegyezik, akkor a szubrutin visszatér, és N-ben van az AZ szám. Ugyanakkor, ha B$-ban a pont van, a keresés befejeződött, és N nullával egyenlő; ez a sikertelenséget jelzi. Eddig minden változatlan.

Most jön a tökéletesítés. Ha a fenti feltételek egyike sem teljesül, akkor a szó még ott lehet valahol a táblázat hátralevő részében. A keresés egy részét megtakaríthatjuk, ha a szó ábécében elfoglalt helyzetét összehasonlítjuk a táblabeli elemével. Ha a keresett szó „D"-vel kezdődik, és az előző táblabeli

```
1080 IFLEN(A$)>5THENA$=LEFT$(A$,5)
1082 A=2:B=1:GOSUB1040
1084 READB$,N:IFB$="."ORB$=A$THENRETURNELSE
1084
```

10-2. ábra. A javított Idword szubrutin
Ügyeljünk arra, hogy a változtatás csak akkor működik, ha a szavak táblázatát ábécésorrendbe szedjük

elem „E"-vel kezdődött, akkor (rendezett táblázatot feltételezve) biztosak lehetünk benne, hogy a további keresés fölösleges. A keresési időben átlagosan 50 százalékot takaríthatunk meg.

Hogyan lehet ezt megvalósítani? A listán látható az IF B$>A$ kifejezés. A Microsoft BASIC-ben a „>" és „<" műveleti jelek felhasználhatók karakterláncok alfabetikus sorrendjének megállapítására. Pl. az „EJTSD" szó ábécésorrendben „kisebb, mint" a NÉZD szó. Tövidített szavaknál a szótári sorrend érvényesül: a „CSEL" megelőzi a „CSELEKEDET"-et, és ezért ábécésorrendben kisebb*.

Tehát, az új Idword egy végső ellenőrzést hajt végre. Ha az utoljára beolvasott táblaelem ábécésorrendben nagyobb a keresett szónál, a szó nem lehet a táblázat hátra levő részében. Az N változóba nullát töltünk, s ez már jelzi a keresés sikertelenségét, így Idword visszatér. Ha belegondolunk, ez hasonló ahhoz, ahogyan arról győződünk meg, hogy egy szó benne van-e a szótárban. Amikor nagyjából ott vagyunk, ahol a szónak lennie kell, egyezést keresünk. Ha részleges egyezést találunk, nem kell tovább keresni. (Képzeljük csak el, mi lenne, ha a szótárak nem lennének ábécésorrendbe szedve!)

Idword-nek ezt a továbbfejlesztett változatát azonnal betehetjük a programba. Ha a szótáblázatot ábécésorrendbe szedjük (dologra fel), a javítás minden zavar nélkül működik. Ne feledkezzünk meg ilyenkor arról, hogy az új szavakat a megfelelő helyre illesszük be!

Súlyos tárgyak

A 6. fejezetben, amely azzal foglalkozik, hogyan változtatja meg a helyszínt a kalandozó, hosszú eszmefuttatás tartozott a Take kezelőhöz. A kezelőt, emlékezzünk csak rá, a VEDD és a LOPD kulcsszó hívja be, ami képessé teszi a játékost a tárgyak felemelésére és továbbcipelésére. Ebben a tekintetben egy sor dolog köti a játékost. Pl. tilos a játékosnak lényeket elmozdítania.

A szállítást azonban leginkább az korlátozza, hogy legföljebb öt tárgyat szabad vinni. A CT(2) változót akkurátusan módosítjuk mindannyiszor, ha egy tárgyat felveszünk vagy elejtünk. A Take kezelő megtiltja újabb tárgy felvételét, ha CT(2) már elérte az ötös maximális értéket.

Ez a fajta korlátozás előnyös abból a szempontból, hogy könnyű betartani, de nélkülöz minden realitást. Végül is a *Kardhalak és kincsek* tárgyai sokfélék — az apró gyűrűtől a nehéz aranykockán át a mázsás súlyú bűvös szekercéig. Biztosan valósághűbb lenne, ha a játékost a tárgyak száma helyett a súlya és terjedelme korlátozná a szállításnál.

*A magyar ékezetes betűk a HT-1080Z-n nem követik az ábécé-sorrendet. Á pl. nem előzi meg B t.

TÁRGY	TÖMEG	TÁRGY	TÖMEG
1 KORONA	25	7 ÉRME	5
2 KOCKA	40	8 HOMOKÓRA	20
3 BOGÁR	15	9 FÁKLYA	15
4 ÖV	20	10 SZEKERCE	20
5 GYÜRÜ	10	11 KULCS	5
6 ÓNIX	10	12 GRÁNÁT	25

10-3. ábra. A táblázat a hordozható tárgyak javasolt tömegszámértékeit tartalmazza

Hogyan érhető ez el? Egyrészt minden tárgyhoz hozzá kell rendelni egy, a hordozhatóság mértékével arányos számot. Miközben a játékos gyűjtögeti a tárgyakat, ezeket a számokat (nevezhetjük őket tömegszámnak) összegezzük és feljegyezzük. Amikor egy újabb tárgy egyedi tömegszámának hozzáadásánál az összeg túllépi az önkényesen megválasztott felső határt, egy üzenet figyelmezteti a játékost, hogy nem veheti fel a tárgyat, mert összerogyna a súly alatt!

A 10-3. ábrán látható a tömegtáblázat; ezen a helyszín minden mozdítható tárgyához egy tömegszámot rendeltünk hozzá. A hozzávetőleges tömegszámok 1 és 50 közé esnek, és értékük egyaránt tükrözi a súlyukat, és hogy milyen könnyen hordozhatók. Nyilvánvaló, hogy ezek önkényesen megválasztott számok. A program leírásai sohasem közlik pl. azt, hogy mekkora a Bűvös gránát vagy hogy miből készült. Az Olvasó ízlésére van bízva, hogy akar-e a számokon változtatni.

Ez volt a dolog egyszerűbb része. Hova tegyük ezeket a számokat, hogy a Take kezelő (ha majd módosítjuk) hozzájuk férjen? Szerencsére számításba vettük ezt. Minden tárgyhoz a tárgyak állapotmátrixának OB(X,0) és OB(X,1) eleme tartozik. OB (X,1) adja meg a tárgy helyét, de OB(X,0) nincs felhasználva. Az egyes tárgyak tömegszámát tárolhatjuk a tárgyak állapotmátrixának fel nem használt elemeiben.

Hogyan kerülnek a tömegszámok a megfelelő változókba? A tárgyak állapotmátrixát a *Kardhalak és kincsek* inicializáló része tölti fel a 3000-es sorban található, tárgyakat inicializáló adatblokkból. Egészen mostanáig az adatblokk minden második eleme kihasználatlan, nulla értékű volt; a másik

```
3000 DATA4,25,7,40,20,15,11,20,5,10,19,10,7
,5,6,20,2,15,3,20,10,5,12,25,4,0,18,0,14,0,
6,0
```

10-4. ábra. A tárgyak módosított inicializáló blokkja: az első 12 tárgynak lesz tömege

adat az egyes tárgyak helyét mutatja a játék kezdetén. Most a nullák helyére írjuk be a megfelelő tömegszámokat.

A 10-4. ábrán látható a 3000-es sor módosított alakja. Az első adat az 1-es tárgy kiinduló helye; az Ékköves korona a 4-es helyiségben található a játék indulásakor. Tömegszáma 25. Amint ezt az adatblokkot beolvassuk, OB(X,0) megvizsgálásával könnyen eldönthetjük, hogy a kérdéses tárgy még felvehető-e.

A Take és a Drop kezelő eléggé megváltozott. Magától értetődik, hogy megszűnt az a régi szisztéma, amelyik CT(2)-ben számlálta a hordozott tárgyakat. Ebből következik, hogy meg kell szüntetni minden szubrutinban minden hivatkozást, amely CT(2)-t eggyel növeli vagy csökkenti.

A 10-5. ábrán láthatók a Take-ben és a Drop-ban elvégzendő változtatások. Lássuk először Take-et, mivel ez változott legtöbbet! A Take-nek most el kell döntenie, hogy nem terheli-e túl magát a játékos, ha az új tárgyat is felveszi. Két dolgot kell feltételezni. Először is, hogy valahányszor egy tárgyat felveszünk, a tömegszámát CT(2)-ben halmozzuk. Másodszor, hogy a hátizsákba összesen 75 tömegszámegységnyi tárgy fér.

A Take eredeti változatában nem számított, melyik tárgyat próbálta felvenni a játékos: mindenképpen tilos volt felvennie, ha a hordozott tárgyak száma ezzel együtt az ötöt meghaladta volna. A régi Take-nek nem kellett kiderítenie, miféle tárgyról van szó, amíg ezt az esetet ki nem zárta. Az új Take-nek azonban tudnia kell, melyik tárgyról van szó, még mielőtt eldöntené, elfér-e még. A 240-es sor a TX$(3)-ban tárolt 2. szó megfejtésével kezdődik. Az Idword szubrutin segítségével megkeressük a tárgy helyét a szótáblázatban, és egyben az N változóban megkapjuk a tárgy AZ számát. Tárgyak esetében az AZ szám megegyezik a tárgy számával.

Ezen információ birtokában a Take elvégezhet egy összehasonlítást. A CT(2)-ben található a játékos által hordozott teljes teher tömegszáma pontokban kifejezve. Ha a tárgy OB(N,0)-ban található tömegének hozzáadása 75-nél nagyobb összeget eredményezne, akkor a játékos már nem bírja el a tárgyat. Ebben az esetben Take úgy folytatódik, mint a korábbi változatban; kiírja a 36-os üzenetet: „KARJAI TELE VANNAK... NEM BÍR EL TÖBBET!" Egyébként, ha a 75 pontos tömeghatárt nem lépi túl, a kezelő

továbbmegy és lehetővé teszi a tárgy felemelését. A teher számontartása miatt CT(2)-t OB(N.0) hozzáadásával megnöveljük.

Most nézzük meg a Drop-ot! Ha az összes előfeltétel teljesül, Drop engedélyezi, hogy a megadott tárgyat eltávolítsuk a hátizsákból. CT(2)-t ismét módosítani kell, hogy a teljes terhet nyilvántartsa. A tárgy OB(X,0)-ban tárolt tömegértékét levonjuk CT(2)-ből.

Ez a javítás más szubrutinokat is érint, pl. a Resur kezelőt, amely üres hátizsákkal éleszti fel a játékost. A többi hasonló javítást is egyszerű elvégezni, a Take-en és Drop-on bemutatott eljárást követve.

Barlangrendszer-változatok

A mikroszámítógépek piaca igazán nem szűkölködik a legkülönbözőbb méretű, változatú és bonyolultságú kalandprogramokban. Ha azonban elkezdjük tanulmányozni a beszerezhető programokat, azt tapasztaljuk, hogy két fő csoport van. Az első csoportba tartoznak a rögzített labirintusú játékok, amilyen pl. a *Kardhalak és kincsek,* ahol a lények és az utak minden játszmában azonosak. Az első típusba tartozó program olyan, mint egy összerakójáték; ha egyszer megoldottuk, a továbbiakban elsősorban azért játsszuk, hogy minél rövidebb idő alatt minél több pontot szerezzünk.

A másik típusba tartozó játékoknál a *labirintus változik,* és a kincsek helyzete, valamint a körülmények minden játszmában különböznek az előzőtől.

```
240 IFCT(2)>=5THENB=36:GOSUB1100:GOTO104:EL
SEA$=TX$(3):GOSUB1080:IFN>9999THENB=7:GOTO2
42:ELSEIFN>12ANDN<17ORN=18THENB=40:GOTO242
241 IFN=17THENB=8:GOTO242:ELSEIFOB(N,1)=21T
HENB=9:GOTO242:ELSEIFOB(N,1)<>CT(0)ORN=0THE
NB=12:GOTO242:ELSEOB(N,1)=21:B=11:CT(2)=CT(
2)+1
242 GOSUB1100:GOTO104
260 A$=TX$(3):GOSUB1080:IFN>9999THENB=7:GOS
UB262:ELSEIFOB(N,1)<>21THENB=10:GOTO262:ELS
EIFN=12THEN540:ELSEOB(N,1)=CT(0):B=11:CT(2)
=CT(2)-1
262 GOSUB1100:GOTO104
```

10-5. ábra. A Take és a Drop kezelők javított változata lehetővé teszi, hogy a tárgyakhoz tömegszámot rendeljünk

Mivel a helyszín elrendezését a véletlen határozza meg, a játék indításakor ezeknek a játékoknak ún. *futási időben kialakuló helyszíne* van.

A változó helyszínnek van néhány hátránya. Először is inkább harc és nem megoldás jellegű. Azaz ritkán találkozunk a játékos eszességét próbára tevő feladatokkal, mivel ezek rendszerint egy bizonyos helyiséghez rögzített ki- és bejáratokhoz kapcsolódnak. (Gondoljunk csak a *Kardhalak és kincsek* keskeny sziklapárkányára!) Másodszor néha háttérbe szorul a helyiségek leírásának szerepe, mivel nagyon nehéz pontos leírást adni olyan átjárókról, amelyek mindig máshol vannak.

Ezek az ellenvetések semmi esetre sem örökérvényűek, és az Olvasó elgondolkodhat azon, vajon nem válna-e a *Kardhalak és kincsek* valamivel érdekesebbé, ha több véletlen tényezőt tartalmazna. Az Olvasó okulására nézzünk néhány módszert arra, hogyan is lehet változó helyszínt létrehozni!

Jelenlegi alakjában programunk nem egykönnyen alakítható át oly módon, hogy az utakat véletlenszerűen állítsuk elő. A helyiségek leírásából derül ki, hogy hol vannak az átjárók. Az összes ilyen jellegű hivatkozást törölni kell. Ezen túl az akadályok funkciója nagymértékben függ attól, milyen irányban hagyja el a játékos a helyiséget vagy lép be oda.

A fentiek miatt, a legjobb, ha úgy képzeljük, hogy a tárgyakat és mindenekelőtt a kincseket véletlenszerűen helyezzük el a helyszín legkülönbözőbb pontjain. Amikor a kalandozó egy játszmába kezd, fogalma sincs róla, hol vannak a kincsek, de később sem, mert játszmáról játszmára változik az is, hogy milyen könnyen találja meg őket.

A 10-6. ábrán látható egysoros kódot betehetjük a program inicializáló részébe. Ez a ciklus az 1 és 8 közötti tárgyakra — a kincsekre — van hatással.

```
12 FORI=1TO8:OB(I,1)=RND(17)+3:NEXTI
```

10-6. ábra. Az inicializáló kód bővítése lehetővé teszi a kincsek véletlenszerű elhelyezését

A kincsek helyzetét, amit később, az OB(I,1) változóban találunk meg, véletlenszerűen jelöljük ki. Az RND(17)+3 kifejezés 3 és 20 közé eső helyiségszámot állít elő. Ez kizárja, hogy a játékos kincset találjon, anélkül, hogy lemenne a föld alatti barlangba; az 1-es és 2-es helyiség ugyanis ki van zárva. (Természetesen ha ezzel nem törődünk, akkor az előzőt helyettesíthetjük az RND(19)+1 kifejezéssel; ez 1 és 20 közé eső helyiségszámot ad.) Sőt az RND(20) kifejezés

lehetővé teszi, hogy egyes kincseket a feledés homályába száműzzünk, a 0-s helyiségbe, ezáltal egy-egy játszmában csökkentjük a maximálisan elérhető pontok számát.

Még egy módon tehetjük a változó helyszínt kockázatosabbá. A fenti példában a FOR-NEXT ciklus határait változtassuk 1-re és 12-re! Ennek hatására véletlen helyre kerül két olyan fontos tárgy, mint a szekerce és a kulcs. Ez néha szörnyű helyzetet teremthet. Pl. a kulcs egy bezárt helyiségbe kerülhet! Aztán a játékosnak esetleg keményen meg kell dolgoznia azért, hogy megtalálja a szekercét és megvédhesse magát. Tegyünk azonban egy szívességet is a szegény játékosnak: tegyük a ciklus mögé az „OB(9,1)=2" utasítást! Ez biztosítja, hogy a fáklya mindig megtalálható a felszínen. Nem várhatjuk el a szánalomra méltó hőstől, hogy vakon tapogatózzon utána a sötétben — vagy igen?

Azon túl, hogy a fenti módon véletlenszerűen jelöljük ki a tárgyak helyét, még egy mód van arra, hogy futás közben alakuljon a *Kardhalak és kincsek* helyszíne. Sok olyan programot írtak, amelyek teljes helyszíneket tárolnak mágnesszalagos adatállományokban. A játékos betölti a fő programot, amelyik viszont betölti a játékos által kiválasztott helyszínt. A végeredmény egy többszintes helyszín, amit csak a szalagon rendelkezésre álló adatállományok száma korlátoz.

Hogyan valósíthatjuk meg ezt az elképzelést? Lényegében a helyszínre vonatkozó összes információ, beleértve a helyiségek és tárgyak leírását, a bejárási táblázatot, az akadályok listáját stb. létrehozható szalagon egy speciális program segítségével. A szótáblázat és az üzenetek blokkja a fő program részei, ezek kezdettől fogva a tárban vannak, de mindezek kiterjesztését mágnesszalagról kell betölteni, hogy a föld alatti helyszín egyes szintjeinek specialitásait támogassák.

Az átalakításnak az a kulcsa, hogy szinte minden, ami most DATA utasításban van, egészen betöltéséig mágnesszalagon van tárolva. A betöltés pillanatában az adatok számjellegű és szöveges változókba töltődnek be. Így tehát egy sor indexes változót kell létrehozni, hogy a mágnesszalagról érkező adatot fogadja. Pl. szintenként 20 helyiség esetén létre kell hozni egy RD$(20,1) mátrixot a helyiségleírások számára. RD$(X,0)-ban van a hosszabb leírás, RD$(X,1)-ben a rövidebb. Egy hasonló OD$(16,1) mátrix kell a tárgyak leírásához, és így tovább.

A bejárási táblázat tömörítése

Igen sok fogás áll rendelkezésünkre, ha a tárfelhasználás csökkentését tűzzük ki célul. Pl. a TRS–80 Microsoft BASIC-ben szereplő PEEK és POKE utasítás segítségével tárterületek érhetők el és változtathatók meg más BASIC struktúrák — pl. indexes változók vagy DATA utasítások — nélkül. Sok esetben

a PEEK és POKE segítségével hatékonyabban tudjuk az adatokat kezelni, mint más módon.

Lássuk ennek legjobb példáját: a bejárási táblázatot! Jelenlegi formájában a bejárási táblázat 20 DATA sorból áll, a helyszín minden helyiségéhez egy sor tartozik. Minden sorban tizenegy elem áll, az elemek értéke 0 és 23 közé esik. Ha összeadjuk az egyes számok által elfoglalt byte-ok számát, valamint hozzáadjuk a DATA utasítások adminisztrálásához szükséges byte-okat, eredményül azt kapjuk, hogy a bejárási táblázat 595 byte-ot foglal el a tárban.

Az elfoglalt hely nagy része kárbavész. Mivel minden adatelem kisebb 256-nál, 1 byte mindegyiknek elég. Ha így áll a helyzet, a teljes tárigény kb. 220 byte: 11 elem megszorozva 20 helyiséggel. A többi helyet olyan adatszerkezetek foglalják el, amelyek csak ahhoz kellenek, hogy az adatelemeket READ paranccsal beolvashassuk. Gondoljuk csak meg: 375 byte-ot fordít a BASIC olyan dolgokra, mint sorszámok, mutatók, DATA kulcsszavak és az elemeket elválasztó vesszők! Ha PEEK-et és POKE-ot használunk, akkor ennek nagy része fölösleges, és 60 százaléknál jóval nagyobb megtakarítást érhetünk el.

Hol lehet elhelyezni ezt a bűvös byte-blokkot, ezt a DATA szerkezetek nélküli adatblokkot? Lehet, hogy meglepően hangzik, de az adatbyte-okat elhelyezhetjük egy BASIC sor belsejében. Mindaddig, amíg ismerjük a BASIC sor pontos kezdőcímét, kedvünk szerint PEEK*-elhetünk a sorban.

Ez a merész kijelentés némi magyarázatra szorul. Az az igazság, hogy a TRS–80 BASIC csak igen kevés megszorítást tartalmaz arra vonatkozóan, hogy mi lehet egy sorban. Az első korlátozás a sor hosszára vonatkozik, és eszerint a sorban legföljebb 255 karakter (vagy byte) lehet. A bejárási táblázathoz 220 elég, tehát ez nem akadály. Másodszor egy BASIC sorban tetszőleges karakter állhat (még különleges vezérlőkarakterek is, amelyek ASCII kódja 32-nél kisebb) — az egyedüli kivétel a nulla kódú karakter. Ennek az az oka, hogy a BASIC a nulla byte segítségével találja meg egy sor végét. Bátran POKE-olhatunk akármilyen értéket a sorba 1 és 255 között, a nulla azonban tilos.

A harmadik kikötés, hogy egy BASIC sor tartalma teljesen értelmetlen lehet a BASIC nyelv szabályai szerint, egészen addig, amíg a program meg nem próbálja a sort végrehajtani. Más szóval teljes zagyvaságot jelentő számokat POKE-olhatunk egy BASIC sorba, ez a programnak meg se kottyan mindaddig, amíg a program végrehajtása a sort elkerüli. Tehát feltölthetünk egy BASIC sort olyan byte-okkal, amelyek csak egy megadott adatkezelő szubrutin számára értelmesek, és nem kell attól tartani, hogy ezzel a BASIC-et összezavarjuk. Még REM utasításra sincs szükség, hogy a sort megvédjük; elég, ha távol maradunk tőle.

*to peek=kukucskálni.

Sürgősen közbe kell még szúrni valamit! Mindezek a megszorítások csak a program helyes futására vonatkoznak. Nem mondtunk semmit a program kilistázásáról. Nyilvánvaló, hogy ha szokatlan byte-okat POKE-olunk egy BASIC sorba, akkor a listán grafikus jelek jelenhetnek meg, sőt a lista teljesen olvashatatlanná válhat, ha vezérlőkarakterek kerültek a sorba, pl. a képernyőtörlés karaktere. Azonban egy szeméttel teleírt lista nem akadálya annak, hogy a program jól fusson. Ha nagyon muszáj, akkor ezeket a furcsa sorokat baj nélkül javíthatjuk is a BASIC EDIT parancsával.

Most már valóban térjünk a lényegre! Az 5000 és 5088 közötti 20 DATA sor helyett egy BASIC sort javaslunk, legyen a sorszáma 5000! Pontos tárbeli címe, hála a program inicializáló részének, úgyis benne van az adatok elérését megkönnyítő DA(X) vektorban. A sor 220 információs byte-ot tartalmaz, ez egyenértékű a bejárási táblázat mostani 220 DATA elemével.

Várjunk csak egy percre! Szinte hallható az ellenvetés. Kezdetben a bejárási táblázat egyes elemeiben nullák vannak, és ez az érték nem fordulhat elő egy BASIC sorban. Másodszor, hogyan is gépelhetnénk ezt be a program írása közben? Nincs megfelelő billentyű az összes 32-nél kisebb ASCII kódú karakter beírására. Mit tehetünk hát?

Mindkét kérdést megoldhatjuk úgy, hogy az adatokat kicsit átkódoljuk. Azt már tudjuk, hogy a bejárási táblázatban található számok 0 és 23 közé esnek. Ha mindegyik értékhez 65-öt hozzáadunk, akkor a számok 65 és 88 közé esnek — 65 és 88 között pedig A-tól X-ig a nagybetűk kódja van. Ezek a betűk ártalmatlanok egy BASIC sorban, és a beírásuk is egyszerű. Valahányszor azonban PEEK utasítással hozzáférünk a bejárási táblázathoz, emlékezzünk rá, hogy 65-öt ki kell vonni az ott talált számból, hogy az eredeti értéket megkapjuk!

Ennyi elég az elméletből; lássuk a kódot! A 10-7. ábrán látható a *Kardhalak és kincsek* két része. Az első maga a bejárási táblázat, átkódolva az 5000-es BASIC sorban tárolva. Hasonlítsuk össze ezt a listát a program mostani változatában található DATA elemekkel, ügyelve arra, hogy az A betű nullát jelent, a B egyet, és így tovább!

A második listán a Travec szubrutin új változata látható. Emlékezzünk rá, hogy jelenlegi formájában, ha a program a bejárási táblázathoz fordul, a D változóba 1 és 11 közötti számot tölt, hogy a táblázat egyik sorából egy adott elemet kiválasszon. (Maga a sor annak a helyiségnek felel meg, amelyikben a játékos tartózkodik; ezt CT(0)-ban tároljuk.) Ezután kerül sor Travec meghívására, ez először megkeresi a megfelelő DATA blokkot, benne a megfelelő sort, kiveszi az elemet, és az A változóba tölti.

A Travec új változatában a bemenő adatok és az eredmény változatlan, de a módszer más. Az új Travec-nek ki kell számítania azt a tárbeli címet, ahol a bejárási vektor keresett byte-ja megtalálható. Az 5000-es sor kezdőcímét már tároltuk DA(1)-ben, ehhez a címhez még egy tényezőt hozzá kell adni, hogy megkapjuk valamelyik byte helyét.

```
5000 BCCBBBBBADJCCCCCCBBCAIJAAEKAAAABAIAFAAL
ADAAAEAAAAAEAAAAFAAAMAAAAAXDAAADAAAAAADAAAA
DAAACAIJAQPJAAJAAHXXXQRRRDAREEAAAAAAAAAAAN
ASAAGAAHAAAAAAMAAAGIAAATAAHATEPAPAPQJAAAAPQ
QAQAKJAABSSSSSSSSSSAMTAAAAAAAAAADAAAASAAOUJW
WWWWWWWTWI

1120 A=(CT(0)-1)*11+(D-1)+DA(1)+4:A=PEEK(A)
-65:RETURN
```

10-7. ábra. Az új, kódolt bejárási táblázat és a Travec szubrutin javított változata
Az 5000-es sort szóközök nélkül kell beírni

Nézzük az 1120-as sort kifejezésről kifejezésre! Először is, bár ténylegesen semmi sem határolja el a byte-okat az új bejárási táblázatban, azok mégis tizenegyes sorozatokba vannak szervezve, egy tizenegyes sorozat helyiségenként. Ha az 1-es helyiséghez tartozó byte-ok a sor elején kezdődnek, a 2-es helyiség byte-jai tizenegy byte-tal később kezdődnek, és így tovább.

A (CT(0)–1)★11 kifejezéssel Travec a mostani helyiséghez tartozó 11 byte-os sorozatnak pontosan a kezdetére lép. Ha az 1-es helyiségről van szó, akkor a kifejezés értéke nulla, ami azt jelenti, hogy az 1-es helyiséghez tartozó byte-ok közvetlenül a sor elején állnak, hozzáadásra nincs szükség. A 2-es helyiség esetében a kifejezés értéke 11, ami azt jelenti, hogy a blokk elejéhez 11-et kell hozzáadni, hogy megkapjuk a 2-es helyiséghez tartozó sorozatot.

Ha már megtaláltuk a megfelelő 11-es sorozatot, hozzáadunk (D–1)-et. Emlékezzünk rá, hogy D épp 1 és 11 közötti szám! A kifejezés 0 és 10 közötti értékre alakítja, ezt kell hozzáadni a sorozat kezdetéhez! Pl. ha az 1. bejárási vektorra van szükség, ez a mostani helyiséghez tartozó sorozat legelső byte-ja, nincs szükség hozzáadásra, vagyis elemátlépésre.

Végül, a DA(1)+4 kifejezés az előzőekhez hozzáadja a 220 byte közül az elsőnek a pontos tárbeli címét. DA(1)-ben van a BASIC sorvektor nevű részének tárbeli címe (pontos leírása a könyv korábbi részében található), a sor tényleges tartalma pedig 4 byte-tal magasabb címen kezdődik. Az eddig összeadott kifejezések egy megadott karakter címét adják eredményül, ezt a címet átmenetileg az A változóban tároljuk.

A tárrekesz tartalmának kiolvasása egyszerű: a PEEK(A) utasítás kiolvassa a rekesz tartalmát és megadja a benne tárolt karakter értékét. Ez az érték 65 és 88 közé esik, nekünk viszont 0 és 23 közötti formában van rá szükségünk. Ezért a PEEK-kel kapott értékből 65-öt kivonunk, és az így kapott eredményt az A változóban tároljuk. Ezzel Travec-nek vége, és visszatér a hívóprogramba.

A felhasználó kétféle hasznot húzhat ebből a megközelítésből. Először is látható a jelentős tárterület-megtakarítás; kb. 370 byte-ot nem lehet csak úgy semmibe venni. Másodszor, a felhasználó valószínűleg értékelni fogja, hogy a mozgási parancsok végrehajtása meggyorsul. Eddig a Travec-nek meg kellett hívnia egy másik szubrutint, hogy az adatmutatót a megfelelő DATA sorra beállítsa, aztán egy FOR-NEXT ciklussal és READ utasításokkal kellett eljutnia a keresett elemig. Most a Travec-nek elég kiértékelnie egy nem túl bonyolult kifejezést, a kiszámított érték segítségével kiolvas egy byte-ot a tárból, az eredményből pedig levon egy rögzített értéket — mindez jóval rövidebb ideig tart, mint a régi módszer FOR-NEXT ciklusai.

Igazság szerint, az 1120-as sor még egyszerűbb is lehetne; azért hagytuk ebben a formában, hogy könnyebb legyen megmagyarázni. Az első rész így is írható: A=(CT(0)-1)*11+D+DA(1)+3.

USR fogások

Az egész könyvben küszködtünk a BASIC-kel, mert a BASIC igen lassú. Van olyan kalandprogram, amelyik átlagosan 20 másodpercet használ el, amíg egy parancsra reagál. Így unalmas a játék, tehát jogos a gyorsítás iránti igény.

A végső elszámolásnál persze a gépi nyelven megírt szubrutinok sokkal gyorsabbak az interpretált BASIC-ben megírtaknál, és az a legjobb kalandprogram, amelyiket teljes egészében gépi nyelven írnak. Nem mindenki hajlandó azonban ennek a feladatnak nekilátni, BASIC-ben sok mindent könnyebb megoldani.

Szerencsére a Microsoft BASIC lehetővé teszi a harmadik megközelítést, nevezetesen a *hibrid programozást,* lehetőséget teremtve rá, hogy alkalmanként a BASIC-ből gyors gépi kódú szubrutinokat hívhassunk meg. Ezt a szolgáltatást a USR(N) utasítással valósították meg. Az utasítás segítségével a BASIC olyan gépi nyelvű szubrutinoknak engedheti át a processzor vezérlését, amelyek a leghatékonyabban oldanak meg bonyolult és sűrűn használt feladatokat.

Az Olvasó már bizonyára többé-kevésbé ismeri a USR(N)-t. Tudja, hogy először a 16526, 16527 címekre kétbyte-os formában be kell POKE-olni a szubrutin kezdőcímét. Tudja, hogy az X=USR(N) kifejezés hogyan használható, és tudja, hogy a kifejezésben szereplő X és N változókkal lehet a behívott szubrutint megváltoztatni.

A USR(N)-nel kapcsolatban gyakran megkérdezik: „Hol helyezzem el a szubrutint?"

A gépi kódú szubrutinokat elhelyezhetjük a tár felső részén, de ebben az esetben a szükséges területet le kell foglalni a MEMORY SIZE kérdésre adott

válasszal*, különben a BASIC karakteres változók részére veszi igénybe. A gépi kódú szubrutint POKE utasítással elhelyezhetjük karakteres változóban is, feltéve, hogy a változót békén hagyjuk. Ez a megoldás nagyon „pocsékoló", mert a programnak tartalmaznia kell egy BASIC szubrutint, hogy a gépi kódot READ-del kiolvassa egy DATA sorból és POKE-kal betöltse a karakterláncba. Végeredményben a szubrutint két példányban kell tárolni: a karakterláncban végrehajtható formában, és a DATA sorban immár haszontalan számok formájában.

Most pedig egy jó hír a USR-t használóknak! (Persze ez nem meglepetés annak, aki elolvasta a fejezet előző részét.) Egy gépi kódú szubrutin elhelyezhető egy BASIC sorban. Ezt a sort a BASIC-nek ki kell kerülnie, egyébként formai hibát okoz: a sort vagy át kell lépni, vagy meg kell védeni az elején elhelyezett REM jelzővel. Végül, a legfontosabbat: nulla byte nem fordulhat elő! Egy rosszul elhelyezett nulla tökéletesen megzavarja a BASIC-et. Szükség van némi körültekintésre, hogy olyan gépi kódú szubrutint írjunk, amiben nincsen nulla, de a feladat megoldható.

A USR(N) használata feltételezi, hogy a szubrutin tárbeli kezdőcímét ismerjük. Egyszerűsíti a dolgot, ha a gépi kódú szubrutint a program első sorában helyezzük el. Ismerve, hogy a BASIC a 17384-es címtől (vagy 17128-tól a Model I* gépeken) kezdve tárolja a programot, kiszámítható a szubrutin kezdő címe. Mivel a program a „RUN" hatására megpróbálná ezeket a sorokat végrehajtani, REM utasításokat kell a sorok elejére tenni, hogy megvédjük őket. A sorszám kódolt formája és a sorvektor összesen négy byte-ot foglal el, valamint a REM jelző is elfoglal két byte-ot**, így a 17391 a megfelelő kezdőcím a gépi kódú szubrutin számára.

A következő kérdés: hogyan juttassuk a szubrutint ebbe a sorba? A válasz egy ideiglenes POKE ciklus. Lapozzunk előre egy pillanatra a 10-8. ábrához! Ezt a rövid kódrészletet használjuk arra, hogy a szubrutint POKE utasításokkal az első BASIC sorba töltsük, ha feltételezzük, hogy az első sor már létezik és elegendően sok szóközt tartalmaz a szubrutin befogadásához. Legvégül a kód törli önmagát, mivel már nincs rá szükség. Az eredményül létrejött BASIC sort mágnesszalagra lehet menteni, ill. onnan be lehet tölteni minden mellékhatás nélkül, ha eltekintünk attól, hogy a LIST parancs meglehetősen bizarr eredményre vezet.

Az eljárás igen egyszerű. A programozó egy REM utasítást ír az 1-es sorszámú sorba és csupa szóközt ír mögé. A szóközök száma attól függ, hogy hány byte-ra van szükség a gépi kódú szubrutinban. Azután beírja a 10-8. ábrán látható sorokat, miután megbizonyosodott arról, hogy ezek a programban az

* HT-1080Z nél a READY?-re adott válasszal.

** Emlékeztetünk rá, hogy a HT-1080Z a TRS-80 Model I-nek felel meg.

***Tapasztalatunk szerint csak 1 byte-ot.

```
20 RESTORE:FORI=17391TO17564:READN:POKEI,N:
NEXT:DELETE20-26:END
22 DATA42,251,64,35,126,35,70,35,254,65,32,
5,120,254,68,40,7,94,35,86,35,25,24,235,30,
9,175,87,25,94,35,86,27,213,42,249,64,126,9
5,35,70,35,78,35,254,3,32,9,120,183,32,5,12
1,254,65,40,5,175,87,25,24
24 DATA231,70,35,94,35,86,225,213,120,254,6
,56,2,6,5,72,65,209,213,126,254,44,40,8,183
,32,8,35,35,35,35,35,35,24,237,254,46,32,2,
225,201,26,19,190,35,40,12,43,35,126,254,44
,40,221,183,40,218,24,245,16
26 DATA214,126,254,44,40,8,183,32,235,35,35
,35,35,35,35,205,90,30,225,213,42,249,64,12
6,95,35,70,35,78,35,254,2,32,9,120,183,32,5
,121,254,78,40,5,175,87,25,24,231,209,115,3
5,114,201

10 CT(0)=1:CT(12)=RND(10)+10:CLS:POKE16526,
239:POKE16527,67

1080 N=0:N=USR(0):RETURN
```

10-8. ábra. Ez a POKE-os programrészlet a gépi kódú szubrutint hozza létre az 1-es sorban
A 10-es sor felkészíti a BASIC-et az USR utasításra, az 1080-as sor meghívja az új szubrutint

első DATA sorok. Amikor leírja, hogy „RUN 20", az 1-es sorban álló szóközök helyét a gépi kódú byte-ok foglalják el, végül a POKE-olást végző kód törli önmagát. Az 1-es sor készen áll rá, hogy USR utasítással meghívjuk.

A szavak kikeresésének meggyorsítása

Ha *valóban* azt akarjuk, hogy gépi kódú programrészek beszúrásával felgyorsítsuk a BASIC-et — melyik funkciókat javítsuk fel? Legtöbb esetben a BASIC sebessége bőven elég, leginkább a parancs kiadása és a válasz között eltelő tetemes időt kell csökkentenünk, hiszen ez a leginkább szembetűnő a BASIC kalandoknál.

Hogyan érhetjük ezt el?

Ha a Microsoft BASIC TRON parancsával nyomon követnénk a *Kardhalak és kincsek* futását, biztosan feltűnne, hogy az egyik szubrutin sohasem akar végetérni. Az Idword szubrutinról van szó, ennek a feladata a beírt szavak

és a szótárban talált szavak összehasonlítása. Ha nem szedtük ábécésorrendbe a szótáblázatot, akkor pusztán annak az eldöntése, hogy egy szó benne van-e a szótárban, eltarthat akár három–négy másodpercig. Sőt, a keresés a lehető leghosszabb ideig tart, valahányszor a beírt szót nem ismeri föl. A játékos azt találja beírni, hogy „OSTOBA JÁTÉK", és hosszú másodpercekig várhat, mielőtt a „MIT MOND?" választ megkapja. Szükségünk van tehát Idword gépi nyelvű változatára. Ez a szubrutin ezredmásodpercekre zsugorítja a táblázat átnézéséhez szükséges időt.

A 10-8. ábrán látható az Idword gépi kódú változata, tárba POKE-olható alakban. Az előzőekből emlékezhetünk rá, hogy egy csupa szóközből álló REM utasítást előre be kell írni, ez fogadja be az információt. Esetünkben a gépi kódú szubrutinnak a 17391-es címtől kezdve 174 byte hely kell, ez szorosan követi a tényleges REM jelzőt a tárban. Ne feledkezzünk meg róla, hogy az 1-es sorban legalább ennyi szóköz legyen, és még egy a REM számára!

Amikor RUN-nal elindítjuk ezt a BASIC szubrutint, az 1-es sor feltöltődik az új szubrutinnal, és a BASIC sorok törlik önmagukat. Az 1-es sorban található furcsa byte-ok semmilyen módon nem zavarják a program betöltését, hogy a LIST parancs hatására az 1-es sor szeméttel tölti meg a képernyőt, de a többi sor listája teljesen normális marad! A 10-8. ábrán látható még néhány változás, amit a program többi részén végre kell hajtani, hogy az új szubrutint befogadja. A *Kardhalak és kincsek* inicializáló részének POKE utasítással a megfelelő értéket kell betöltenie a USR mutatójába, hogy a USR meghívása a 17391-es tárcímre adja a vezérlést. Másodszor, Idword jelenlegi változatát ki kell cserélni az itt látható egyszerűbb sorra. Figyeljük meg, hogy az N változót nullára állítjuk be! Ez biztosítja az N változó létezését, hogy az új szubrutin meg tudja találni és képes legyen kezelni. Ily módon a szubrutint nem kell felkészíteni arra a meglehetősen bonyolult műveletre, hogy új változót hozzon létre.

(A 10-8. ábrán látható értékek közül egyesek különböznek a TRS–80 Model I felhasználói számára. Ennek az az oka, hogy a Model I kb. 256 byte-tal kisebb címen kezdi a BASIC program tárolását, mint a MODEL III. Ezért a gépi kódú szubrutint tároló POKE ciklusnak a 17135-ös címmel kell kezdenie 17391 helyett. Hasonlóan, a USR mutató beállításakor a 239 és 66 számokat kell tárolni 239 és 67 helyett. Magának a szubrutinnak a kódja nem változik; az ugyanis áthelyezhető, más néven címtől független.)

Azok a BASIC felhasználók, akik még sohasem kóstoltak bele a gépi nyelvbe, anélkül is használhatják a szubrutint, hogy törődnének azzal, hogyan is működik. A „merészebbek" számára azonban úgy tisztességes, ha közreadjuk a szubrutin forrásnyelvű listáját, a 10-9. ábrán látható rövid magyarázat kíséretében.

Először is lássuk, milyen követelményeknek kell eleget tennie a szubrutinnak, amit továbbra is Idword-nek hívunk! A hívóprogram az A$ szöveges változóban tárolja a keresendő szót. Az Idword egyesével összehasonlítja ezt a szót a szótáblázat elemeivel, amíg csak egyezőt nem talál vagy el nem éri a

10-9. ábra. Az új, gépi kódú idword forrásnyelvű listája

```
40F9                VARIAS  EQU     16633
40FB                ARRAYS  EQU     16635
42EF                        ORG     17135
42EF 2AFB40         IDWRD:  LD      HL,(ARRAYS)
42F2 23             ID1:    INC     HL
42F3 7E                     LD      A,(HL)
42F4 23                     INC     HL
42F5 46                     LD      B,(HL)
42F6 23                     INC     HL
42F7 FE41                   CP      65
42F9 2005                   JR      NZ,ID2
42FB 78                     LD      A,B
42FC FE44                   CP      68
42FE 2807                   JR      Z,ID3
4300 5E             ID2:    LD      E,(HL)
4301 23                     INC     HL
4302 56                     LD      D,(HL)
4303 23                     INC     HL
4304 19                     ADD     HL,DE
4305 18EB                   JR      ID1
4307 1E09           ID3:    LD      E,9
4309 AF                     XOR     A
430A 57                     LD      D,A
430B 19                     ADD     HL,DE
430C 5E                     LD      E,(HL)
430D 23                     INC     HL
430E 56                     LD      D,(HL)
430F 1B                     DEC     DE
4310 D5                     PUSH    DE
4311 2AF940                 LD      HL,(VARIAS)
4314 7E             ID4:    LD      A,(HL)
4315 5F                     LD      E,A
4316 23                     INC     HL
4317 46                     LD      B,(HL)
4318 23                     INC     HL
4319 4E                     LD      C,(HL)
431A 23                     INC     HL
431B FE03                   CP      3
431D 2009                   JR      NZ,ID5
431F 78                     LD      A,B
```

```
4320 B7       04100        OR     A
4321 2005     04200        JR     NZ,ID5
4323 79       04300        LD     A,C
4324 FE41     04400        CP     65
4326 2805     04500        JR     Z,ID6
4328 AF       04600 ID5:   XOR    A
4329 57       04700        LD     D,A
432A 19       04800        ADD    HL,DE
432B 18E7     04900        JR     ID4
432D 46       05000 ID6:   LD     B,(HL)
432E 23       05100        INC    HL
432F 5E       05200        LD     E,(HL)
4330 23       05300        INC    HL
4331 56       05400        LD     D,(HL)
4332 E1       05500        POP    HL
4333 D5       05600        PUSH   DE
4334 78       05700        LD     A,B
4335 FE06     05800        CP     6
4337 3802     05900        JR     C,ID7
4339 0605     06000        LD     B,5
433B 48       06100 ID7:   LD     C,B
433C 41       06200 ID8:   LD     B,C
433D D1       06300        POP    DE
433E D5       06400        PUSH   DE
433F 7E       06500 ID9:   LD     A,(HL)
4340 FE2C     06600        CP     44
4342 2808     06700        JR     Z,ID10
4344 B7       06800        OR     A
4345 2008     06900        JR     NZ,ID11
4347 23       07000        INC    HL
4348 23       07100        INC    HL
4349 23       07200        INC    HL
434A 23       07300        INC    HL
434B 23       07400        INC    HL
434C 23       07500 ID10:  INC    HL
434D 18ED     07600        JR     ID8
434F FE2E     07700 ID11:  CP     46
4351 2002     07800        JR     NZ,ID11A
4353 E1       07900        POP    HL
4354 C9       08000        RET
```

```
4355 1A        08100 ID11A:  LD      A,(DE)
4356 13        08200         INC     DE
4357 BE        08300         CP      (HL)
4358 23        08400         INC     HL
4359 280C      08500         JR      Z,ID13
435B 2B        08600         DEC     HL
435C 23        08700 ID12:   INC     HL
435D 7E        08800         LD      A,(HL)
435E FE2C      08900         CP      44
4360 28DD      09000         JR      Z,ID9
4362 B7        09100         OR      A
4363 28DA      09200         JR      Z,ID9
4365 18F5      09300         JR      ID12
4367 10D6      09400 ID13:   DJNZ    ID9
4369 7E        09500         LD      A,(HL)
436A FE2C      09600         CP      44
436C 2808      09700         JR      Z,ID14
436E B7        09800         OR      A
436F 20EB      09900         JR      NZ,ID12
4371 23        10000         INC     HL
4372 23        10100         INC     HL
4373 23        10200         INC     HL
4374 23        10300         INC     HL
4375 23        10400         INC     HL
4376 23        10500 ID14:   INC     HL
4377 CD5A1E    10600         CALL    1E5AH
437A E1        10700         POP     HL
437B D5        10800         PUSH    DE
437C 2AF940    10900         LD      HL,(VARIAS)
437F 7E        11000 ID15:   LD      A,(HL)
4380 5F        11100         LD      E,A
4381 23        11200         INC     HL
4382 46        11300         LD      B,(HL)
4383 23        11400         INC     HL
4384 4E        11500         LD      C,(HL)
4385 23        11600         INC     HL
4386 FE02      11700         CP      2
4388 2009      11800         JR      NZ,ID16
438A 78        11900         LD      A,B
438B B7        12000         OR      A
```

```
438C 2005         12100        JR      NZ,ID16
438E 79           12200        LD      A,C
438F FE4E         12300        CP      78
4391 2805         12400        JR      Z,ID17
4393 AF           12500 ID16:  XOR     A
4394 57           12600        LD      D,A
4395 19           12700        ADD     HL,DE
4396 18E7         12800        JR      ID15
4398 D1           12900 ID17:  POP     DE
4399 73           13000        LD      (HL),E
439A 23           13100        INC     HL
439B 72           13200        LD      (HL),D
439C C9           13300        RET
0000              13400        END
00000 TOTAL ERRORS
```

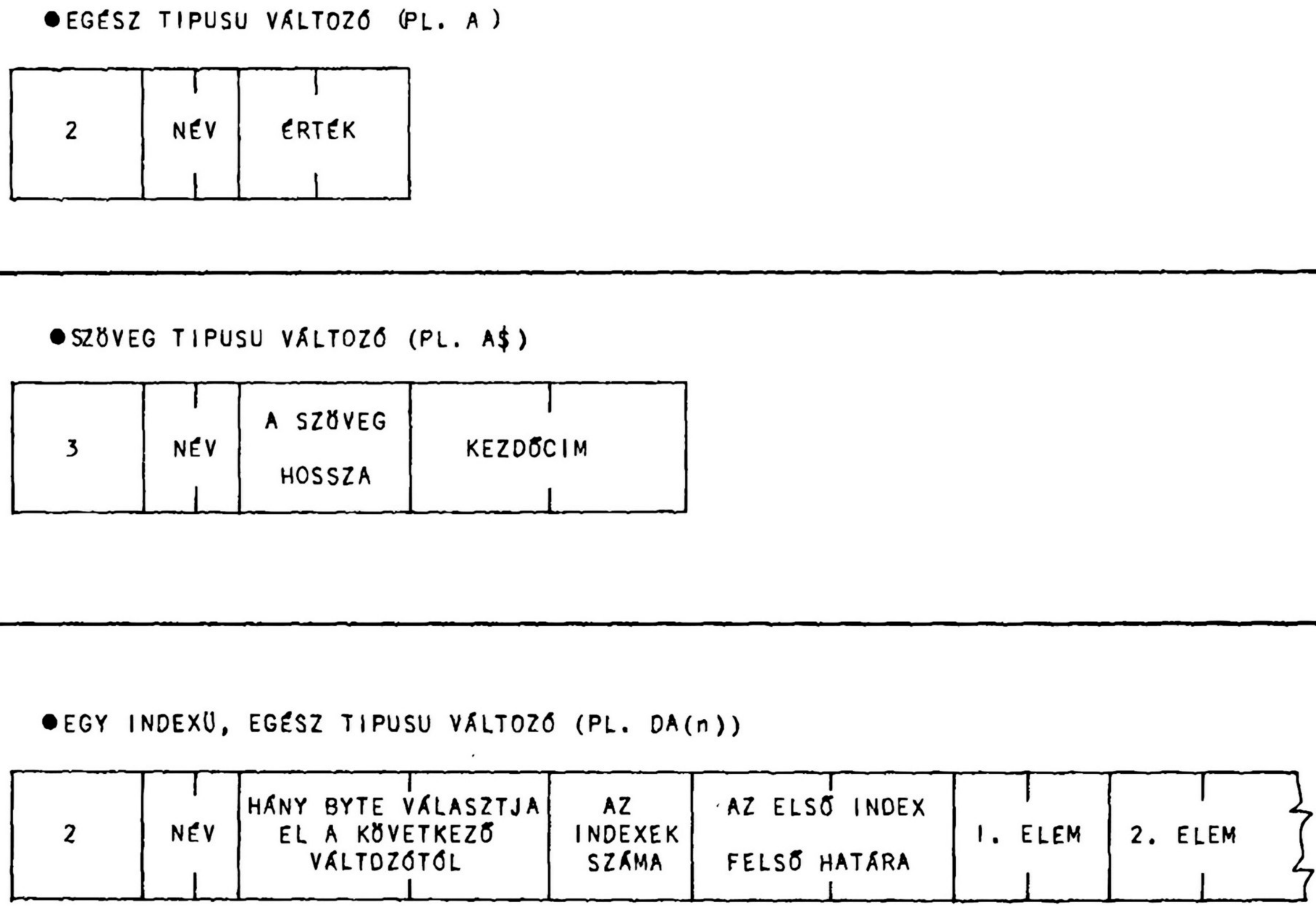
táblázat végét. Ha megtalálta a szót, akkor beolvassa a hozzá tartozó AZ számot, és az N változóban tárolja; sikertelen keresésnél N értéke 0 lesz.

Egy táblázatban kereső gépi kódú szubrutint nem nehéz megtervezni. A feladatnak az az elgondolkodtató része, hogyan kapcsolódjunk a változókhoz. Hol vannak a tárban? Milyen a formátumuk? Hogyan találjuk meg és változtatjuk meg értéküket?

A dolgok megértését megkönnyítik a 10-10. ábrán látható diagramok. Kétfajta tárbeli adatszerkezettel lesz dolgunk: egyszerű változókkal, mint pl. N és A$, valamint indexes változókkal, pl. DA(*n*)-nel. Ezeket az adatszerkezeteket a BASIC a tényleges programsorokat követő szabad tárterületen tárolja. Elöl állnak minden rendezettség nélkül az egyszerű változók; őket követik az indexes változók. A BASIC két mutatót használ, hogy megkönnyítse az adatszerkezeteket tároló terület megtalálását. A 16633 és 16634 című tárolók tartalmazzák az egyszerű változók kezdőcímét. Ezt a mutatót Varias-nak nevezzük el. A 16635, 16636 című rekeszek az indexes változók kezdetére mutatnak, ezt a mutatót Arrays-nek nevezzük.

Figyeljük meg, hogy a szövegek nem közvetlenül azon a területen helyezkednek el, ahol a többi változó található! Azon a területen csak egy mutatót találunk, ez adja meg a szöveg címét. A szövegeket a gép magas címeken, a rendelkezésre álló tár vége felé tárolja.

Az új Idword első dolga, hogy megkeresse a szótáblázat kezdetét. Ez a cím a DA(2)-ben található, mivel a szótáblázat a 2. DATA blokk. Idword az Arrays mutatóból betölti az indexes változók területének kezdőcímét. Ezután



10-10. ábra. Idword ezt a három típusú változót kezeli
Az egész változókat és az indexes változókat a gép külön RAM területen tárolja

egy olyan bejegyzést keres, amelynél a második byte-on A betű, a harmadik byte-on D betű áll, mivel a nevet a gép fordított sorrendben tárolja. Ezeknek a betűknek az ACSII kódja 65 és 68. A negyedik és ötödik byte-ban tárolt érték segítségével átlépjük azokat az indexes változókat, amelyeknél a név nem egyezik a kereséttel, ugyanis ez a két byte tartalmazza, hány byte tartozik még a változóhoz. Az Idword végül megtalálja DA(*n*)-t.

Miután a tömböt megtalálta, ki kell keresnie DA(2) értékét. Az indexes változó elemeinek tényleges értékei a név után öt byte-tal kezdődnek, az első elem DA(0). Mivel minden érték két byte-ot foglal el, DA(2) összesen kilenc byte-tal a név után kezdődik. Tehát Idword ennyit előrelép, kiolvassa a két byte-os értéket, és egy PUSH utasítással a verembe tolja.

Most szükségünk van A$ tárbeli kezdőcímére, hogy az összehasonlításokat elvégezhessük. A Varias mutató segítségével a szubrutin újabb keresésbe kezd két ismérv szerint: az első byte legyen három, ez jelzi a szöveg típusú változót, a második és harmadik byte pedig legyen nulla, ill. 65, ez felel meg az A névnek. (Egybetűs változónév esetén maradék byte-ba 0 kerül.) Ha a bejegyzés nem felel meg ezeknek a követelményeknek, akkor átlépjük. Annyi byte-ot lépünk át, amennyi az első byte-nak, a típusazonosítónak az értéke. A 2-es típusú változók, az egészek, két byte-on tárolnak egy változót, a 3-as típusú — azaz karakter típusúak — három byte-ot használnak arra, hogy a szövegre mutassanak. Tehát az azonosítószámból Idword megtudja, mennyit kell átlépnie, hogy a következő változót megtalálja. (Mivel a BASIC programban a DEFINT utasítás van, csak erre a két típusú változóra kell számítani, egyszeres vagy kétszeres pontosságú változók nincsenek.)

Amikor az A$ változót megtaláljuk, a negyedik byte értékét megjegyezzük; ebből derül ki, milyen hosszú A$. A következő két byte-ban található a karakterlánc tárbeli kezdőcíme; ezt is tároljuk. Végül az A$ karakterlánc B regiszterben tárolt hosszát módosítjuk; úgy, hogy 5-nél ne legyen nagyobb. Ily módon bármilyen beírt szónak csak az első öt betűje lényeges. A helykímélés miatt a szótáblázat elemei legföljebb öt karakter hosszúak. A karakterlánc kezdőcímét egy PUSH utasítással biztonságba helyezzük a veremben. A lánc hosszát, későbbi felhasználás céljából, a C regiszterben rejtjük el.

Most már komolyan belefogunk a táblázat átnézésébe. Előkészületként Idword a B regiszterbe tölti A$ hosszát, DE-be kerül A$ kezdőcíme. Ha azt tartja számon, hogy hol tartunk a tárban; a táblázat kezdetétől a végéig növekszik. Ezután elkezdődik az összehasonlító ciklus. Idword ellenőrzi, hogy a szótáblázat következő karaktere nulla vagy vessző-e (ASCII kódja 44). Ha igen, akkor egy DATA elem végéhez értünk, meg kell keresni a következő elemet. Ha nullába ütköztünk, akkor a DATA sor végéhez értünk, és Idword-nek 5 byte-ot kell átlépnie, hogy a következő sor elejét elérje. Ha vesszőt találtunk, akkor egy byte-ot (magát a vesszőt) kell átlépni, hogy a következő elemet megkapjuk. Ezután a szubrutin visszatér a ciklus elejére, hogy B és DE beállításával előkészítse a következő összehasonlítást. Mindez azért van, mert a most

folyó összehasonlítást sikertelennek tekintjük, ha korán érjük el egy DATA elem végét.

Ha egyik elem végét jelző kódot sem észleltük, akkor még egy előzetes ellenőrzést végzünk. Ha pontot (ASCII kódja 46) találunk, akkor az egész keresés sikertelen. Miért? Azért, mert a táblázat utolsó szava egy pont; ha ezt megtaláltuk, akkor a keresés elérte a táblázat végét, anélkül, hogy egyezést találtunk volna. Az Idword POP utasítással rendbehozza a vermet, aztán visszatér. N nullával egyenlő, jelezvén a keresés kudarcát, mivel N-t közvetlenül a USR hívása előtt beállítottuk nullára.

Ha a fenti kódok egyikét sem találja, akkor Idword végrehajtja a szavak tényleges összehasonlítását. A$ egyik betűjét, erre DE mutat, összehasonlítja a szótáblázat megfelelő betűjével, amelyre HL mutat. Ha az összehasonlítás sikertelen, akkor lefut egy rövid ciklus, amely átlépteti HL-t a táblabeli elem maradékán, az elem végét jelző nulláig vagy vesszőig. Ezután Idword visszatér a ciklus elejére, hogy DE-t és B-t ismét beállítsa egy új összehasonlításhoz.

Ha azonban a két betű egyezik, akkor B-t eggyel csökkentjük, és a következő betűket hasonlítjuk össze. Ez a ciklus addig folytatódik, míg B nullára nem csökken. Amikor ez bekövetkezik, Idword tudja, hogy A$ minden betűje megtalálható-e a táblázatnak ebben az elemében. Ez azonban nem elég; mi történjen, ha A$ csak egy része a táblabeli elemnek? (Pl. ha azt írjuk le, hogy D, ez kiállja a fenti összehasonlítást a DK táblabeli elemmel, de a kettő mégsem ugyanazt a parancsot jelenti.) Még egy ellenőrzést kell elvégezni. A táblázatbeli elem következő karakterét is megvizsgáljuk. Ha az elem végét jelző nulla vagy vessző következik, akkor az egyezés teljes. Ellenkező esetben Idword átugrik arra a ciklusra, amelyik átlépi az elem hátralevő részét és új elemet keres.

Ha az egyezés teljes, akkor az utolsó feladat az, hogy beolvassuk a szótáblázat rá következő elemét, vagyis az AZ számot, és ezt a számértéket az N változóban tároljuk. A talált elem végétől függően Idword továbblép a következő elemhez. Ezután meghívja a TRS–80 ROM-jában az 1E5AH címen kezdődő szubrutint. A szubrutin bemenő adata egy ASCII-ben kódolt szám, amelynek kezdőcíme a HL regiszterpárban van. Ezt az értéket átalakítja kettes számrendszerben ábrázolt egésszé és DE-ben tárolja. Ha ezzel végeztünk, akkor a vermet beállítjuk, és DE tartalmát átmenetileg beletesszük.

Befejezésül Idword-nek meg kell találnia az N változót a tárban, hogy megváltoztassa értékét a kettes számrendszerben kódolt AZ számra. Hasonló módon keressük, mint korábban A$-t. Mikor N-t megtaláljuk, az értéket kivesszük a veremből, és a változó negyedik és ötödik byte-jába töltjük. Idword ezzel befejeződött, visszatér tehát az USR hívást lezárva.

A szubrutin tartalmaz egy bizonyos következetlenséget, amely azonban nem befolyásolja működését, ugyanis a táblázat összes elemét megvizsgálja — az AZ számokat is beleértve. (Egy korrekt programnak át kellene lépnie őket.) Ha a játékos szó helyett számot ír le, akkor fennáll annak a lehetősége, hogy az megegyezik egy AZ számmal. De ha be is következik ez a hibás egyezés,

semmi baj nem történik. Miért nem? Amikor egyezést talál, akkor ldword a következő elem számszerű értékét betölti N-be, abban a hiszemben, hogy az egy AZ szám. Mivel azonban a következő elem valójában nem számértéket tartalmaz, értékéül (az 1E5AH címen kezdődő ROM szubrutinból) nullát kapunk. Tehát a hibás egyezés eredményeként N mindig nulla lesz, ami a hívóprogram számára azt jelenti, hogy nem volt elfogadható egyezés.

Megéri-e a fáradságot, hogy beépítsük ezt a szubrutint a programba? Próbálja ki az Olvasó, maga is meglátja! Mivel a parancs beírásától a válaszadásig eltelt időnek legalább 80 százaléka a szótáblázat vizsgálatával telik el, a gépi kódú ldword beépítésével nyert sebességnövekedés bőven megéri azt a kis bonyodalmat, amit az új 1-es sor létrehozása és a 10-8. ábrán látható egyszerű POKE ciklus beírása jelent.

Befejezésül

A legtöbb Olvasó számára a hibrid *Kardhalak és kincsek* sebessége bőven elég ahhoz, hogy élvezetes legyen a kalandozás. Egy-két megszállott programozó további pontosításra és hatékonyságnövelésre vágyik. Ezek az Olvasók végül megkísérlik a legvégső megoldást — egy teljesen gépi kódban megírt kalandprogramot. Ezeknek az emelkedett lelkeknek segítünk néhány megjegyzéssel és tanáccsal.

Mindenekelőtt a gépi kódú kalandban rejlő kihívást még nehezebbé teszik a berendezés korlátai. Emlékezzünk rá, hogy a könyvben azt tételeztük fel, hogy az Olvasónak mindössze egy 16K tárral és mágnesszalagos beviteli/kiviteli berendezéssel rendelkező TRS–80 gépe van, mágneslemez nélkül! Ezzel az a baj, hogy a mágnesszalagot használó szövegszerkesztők és assemblerek gyakorlatilag használhatatlanok nagyobb méretű programok írására. Mágnesszalagos assemblerrel nem célszerű olyan programot írni 16K-s gépen, amelynek mérete lefordítva 1K-nál nagyobb.

Ezt a korlátozást úgy kerülhetjük meg, hogy a programot lefordított modulok sorozatából hozzuk létre. A csapda abban rejlik, hogy egyik modul sem tudja, milyen címeket használjon, amikor egy másik modulra hivatkozik, hacsak nem adjuk meg nyíltan ezeket a címeket minden egyes modulban. Ebben az esetben azonban, ha valamelyik modulban változtatunk, ez maga után vonja, hogy rögtön sok modulban kell változtatni. (A lemezt használó fordítók ún. szerkesztőprogrammal oldják meg ezt a problémát, amely összeszerkeszti a modulokat, miközben kitölti az egyik modul hivatkozásait a másik modulban szereplő címekre.)

Tegyük fel, hogy az Olvasó hajlandó elviselni ezeket a kényelmetlenségeket! A következő probléma, amivel meg kell birkóznia, adminisztratív jellegű. A BASIC egyik szépsége abban áll, hogy a PRINT segítségével nagyon könnyű több sor szöveget is kiírni. Mit tehetünk gépi kódban? Hogyan írathatunk ki

CIM	
032AH	Az A-ban tárolt karaktert kiirja a képernyőre
28A7H	A képernyőre kiirja azt az üzenetet, amelyiknek a végén nulla áll, kezdőcime pedig a HL regiszterben van
1BB3H	A billentyűkről beolvas legföljebb 255 karaktert, és ezt egy pufferben tárolja. Hiváskor HL legyen eggyel kisebb a puffer kezdőciménél
0049H	A szubretin megvárja, mig lenyomnak egy billentyűt. A lenyomott billentyű kódját az A regiszterbe tölti

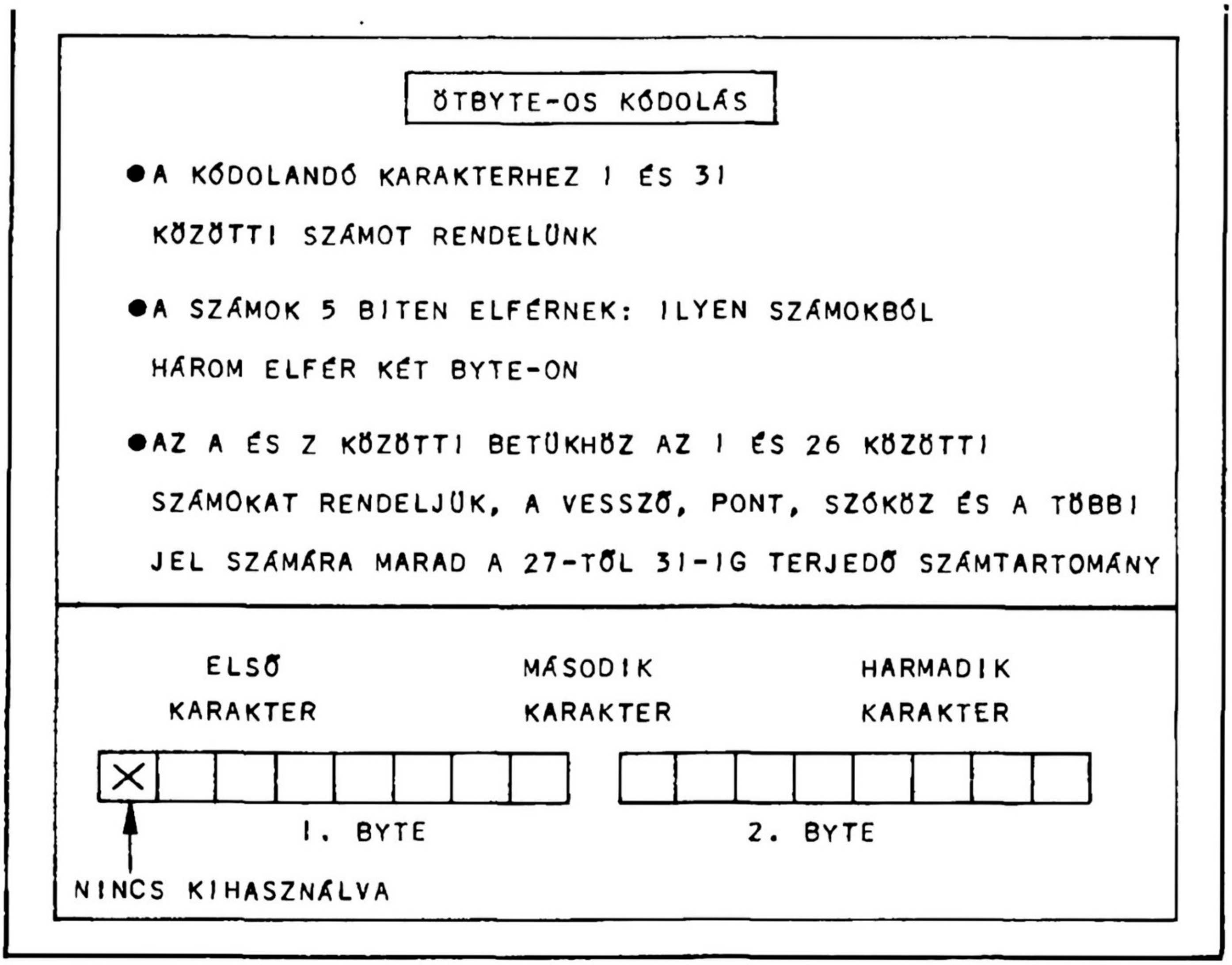
10-11. ábra. A gépi kódú kalandprogramból hivható néhány ROM szubrutin címe

egy sort? Vagy hogyan olvassunk be egy parancsot a billentyűkről? Szerencsére a TRS–80 ROM-jában elég sok szubrutint találunk, amelyek alkalmasak ezeknek a kiszolgálófeladatoknak az elvégzésére. A 10-11. ábrán néhány példát látunk erre.

Az adminisztrációs feladatok másik része a változókkal van kapcsolatban. BASIC-ben leírhatjuk, hogy A = 1. Nem a mi feladatunk, hogy a tárban helyet keressünk az A változónak — a BASIC ezt megteszi. A gépi nyelven programozható kalandozónak előre kell gondolkodnia, és a tárban területeket kell fenntartania a kalandprogramban szükséges számok tárolására. Meg kell írni a fenti területeket kezelő szubrutinokat.

Nézzük viszont az előnyöket! Mivel a gépi kód igen gyors, igénybe vehetünk olyan fogásokat, amelyek tárterületet takarítanak meg; ezek a lassú BASIC esetében nem lennének célszerűek. A legjobb fogást nevezhetnénk „tömörített leírásnak".

Mi a tömörített leírás? Először is gondoljunk arra a tárterületre, amit a BASIC kalandprogramban a helyiségek és a tárgyak leírása foglal le. Ha ezeket a sorokat valamilyen tömörített formában tudnánk tárolni, ezzel sok helyet



10-12. ábra. Egy lehetséges módszer karakterek tömörített tárolására
Bár a módszer nem túl hatékony, gépi nyelven könnyen megvalósítható

takarítanánk meg. A tömörített leírásban a szöveget kódolt formában tároljuk, és csak akkor dekódoljuk, amikor ki kell írni.

A 10-12. ábrán látható, hogyan tárolhatunk egy szöveget tömörített formában; sok más módszer is ismeretes. Ennél a módszernél a szövegben csak 31 különböző jel lehet: a 26 betű és néhány elválasztókarakter, pl. a szóköz és a pont. Egy bekezdést úgy kódolunk, hogy a karaktereket hármasával a tár két byte-jába sűrítjük össze úgy, hogy minden karakterhez 1 és 31 közötti értéket rendelünk. Mivel a fenti tartományba eső számok öt biten elférnek, három kódolt karakternek 15 bit kell, ami könnyedén befér két byte-ba. Tehát, ha lemondunk néhány jelről, 33 százalékos helymegtakarítást érhetünk el.

11. FEJEZET

Grafikus kalandok: alapfogalmak

Miután az utolsó oldalakon is túljutott az Olvasó, azt hiheti, hogy a kalandprogramok írásának témáját kimerítettük. Ám egyre növekszik azoknak a száma a mikroszámítógépek felhasználói táborában, akik vitába szállnak ezzel az állítással. Végül is az eddig bemutatott szöveges típusú kalandprogramok tulajdonképpen a ma oly divatos számítógépes játékok ősei. Helyettük a kalandprogramok újabb típusai árasztják el a piacot, kiszorítva a régieket. Az újabb játékok döntő többsége grafikus kalandprogram.

Mi ennek az oka? Az egyik ok az, hogy ki kell használni a korszerű gépek felépítését. Régebben a számítógépet időosztásos alapon, terminál mellett használták, és a terminál nem is volt okvetlenül képernyő, esetleg csak egy elektromechanikus írógép. A kalandok szükségszerűen szövegesek voltak. A mai, kis számítógépek viszont döntő többségükben jó minőségű grafikával rendelkeznek, és a kurzor is könnyen irányítható. Amikor a mikroszámítógépes játékok zöme kihasználja a képernyő nyújtotta sokoldalú lehetőségeket, miért épp a kalandprogramok ne tennék?

A kalandok összehasonlítása

A szövegekre alapozott és a grafikus kalandok mind felépítésükben, mind működésükben több vonatkozásban eltérnek. Ha az Olvasó emlékszik a korábbi fejezetekre, maga is rájöhet néhányra. Íme néhány eltérés:

Először is nem olyan nagy a tárigény. Mi fogyasztotta igazából a tárat az előző típusú játéknál? Napnál világosabb, hogy szöveg foglalta a helyet: a helyiségek és a tárgyak leírása, a különleges üzenetek és a szavak táblázata. A játék grafikus változatában lényegesen kevesebb szövegre van szükség. A helyiségek leírását teljesen felváltja a helyiség rajza. Hasonlóan a tárgyak leírását a képernyőn megjelenő jellegzetes karakterek váltják föl; ezek jelzik a tárgyakat.

Kevesebb különleges üzenetre van szükség. A parancsokat egy-egy billentyű lenyomásával adjuk ki, ez szükségtelenné teszi a szótáblázatot és az elemző szubrutinokat.

Nyilvánvaló, hogy a megnövekedett szabad tár több helyiséget enged meg. A grafikus kalandokban általában jóval több helyiség van, mint a versenytárs szöveges kalandprogramokban. Pl. hasonlítsuk össze a *Kardhalak és kincsek*-et a példának választott grafikus játékkal, a *Szörnyek az útvesztőben*-nel! Az első játékban 20 helyiség van, a másodikban összesen 90! A sok-sok helyiség feltétlenül megnöveli a kalandjáték érdekességét.

Másodszor a grafikus kalandot valós időviszonyok között játsszák. Más szóval a másodpercek múlása igen fontos tényező. A régebbi játékokat paranccsal vezérelték; az eseményeket a beírt parancsok váltják ki, de a következő parancs beírásáig minden mozgás megáll. Az újabb játékokat az idő múlása vezérli; a cselekmény az idővel együtt halad, és bármikor úgy dönthetünk, hogy beavatkozunk.

Ez ismét az érdekességet növeli. Ha a grafikus játékban egy lény támadásba lendül, nem lehet egyszerűen félrevonulni egy kis tízórai szünetre — küzdened kell vagy meghalsz! Hangsúlyt kap egy új tényező, a gyors szem/kéz reakció.

Harmadszor, megnövekszik a küzdelem jelentősége. A régebbi játékokban a lények nagyrészt a mozgást akadályozták, a néha felbukkanó kitartó lény kivételével. A grafikus játékokban az összes lény kitartó, ellenséges, és életre-halálra küzd a kalandozóval. A *Kardhalak és kincsek*-ben a küzdelem eredményét véletlen számok döntötték el; a *Szörnyek az útvesztőben* játékban az ellenfelek erejét számontartjuk, ez határozza meg az összecsapás kimenetelét.

Tehát az új játékosnak bölcs harcosnak kell lennie! Győzelme most már nem a számítógép által feldobott érmétől függ. Tekintetbe kell vennie a rendelkezésére álló fegyvereket, a saját erejét, hogy meddig tart ki az élelem és a gyógyszer, és nem utolsósorban az ellenfél erejét.

Negyedszer: a helyszín részletei a véletlentől függően a játék elején alakulnak ki. A legtöbb tárgy helyzete a játék futásának kezdetén dől el, nem a program megírásakor. Más szóval, míg a helyiségek térképe és a helyiségek részletei változatlanok maradnak, a kincsek és lények elhelyezése minden játékban más és más.

Végül, a struktúra szempontjából nézve, a grafikus kalandokat könnyebb elkészíteni. A vezérlőnek nem kell a parancsok sok lehetséges változatát értelmeznie. A kevesebb parancs kevesebb kezelőt jelent. Mivel a tárgyakat speciális karakterek jelzik a képernyőn, a képernyőt magát is tekinthetjük információtárolónak; ezért kevesebb adatot kell tömbökben tárolni.

Van egy eset, mikor ez az általános előny nem érvényesül; ez a tényleges képernyőkezelés. A *Kardhalak és kincsek*-ben elég volt PRINT utasításokat használni. Most azzal is törődni kell, hogyan ábrázoljuk grafikusan a tárgyakat, a lényeket és a kincseket. Gondosan kell ügyelni a mozgatásukra, és tisztában kell lenni azzal is, mi történik, ha két mozgó objektum útja keresztezi egymást.

Kell írni egy képernyő-felfrissítő szubrutint, amelynek azonban nem áll rendelkezésére két perc ahhoz, hogy felrajzoljon egy újonnan felkeresett helyiséget. A szöveges kalandokban a szövegkezelés bajnokai lettünk (parancsok elemzése, keresés a szótáblázatban, szövegek kiolvasása), a rajzos játéknál a grafika mestereivé kell válni (mit POKE-oljunk, hova POKE-oljunk).

Egy helyiség megjelenítése

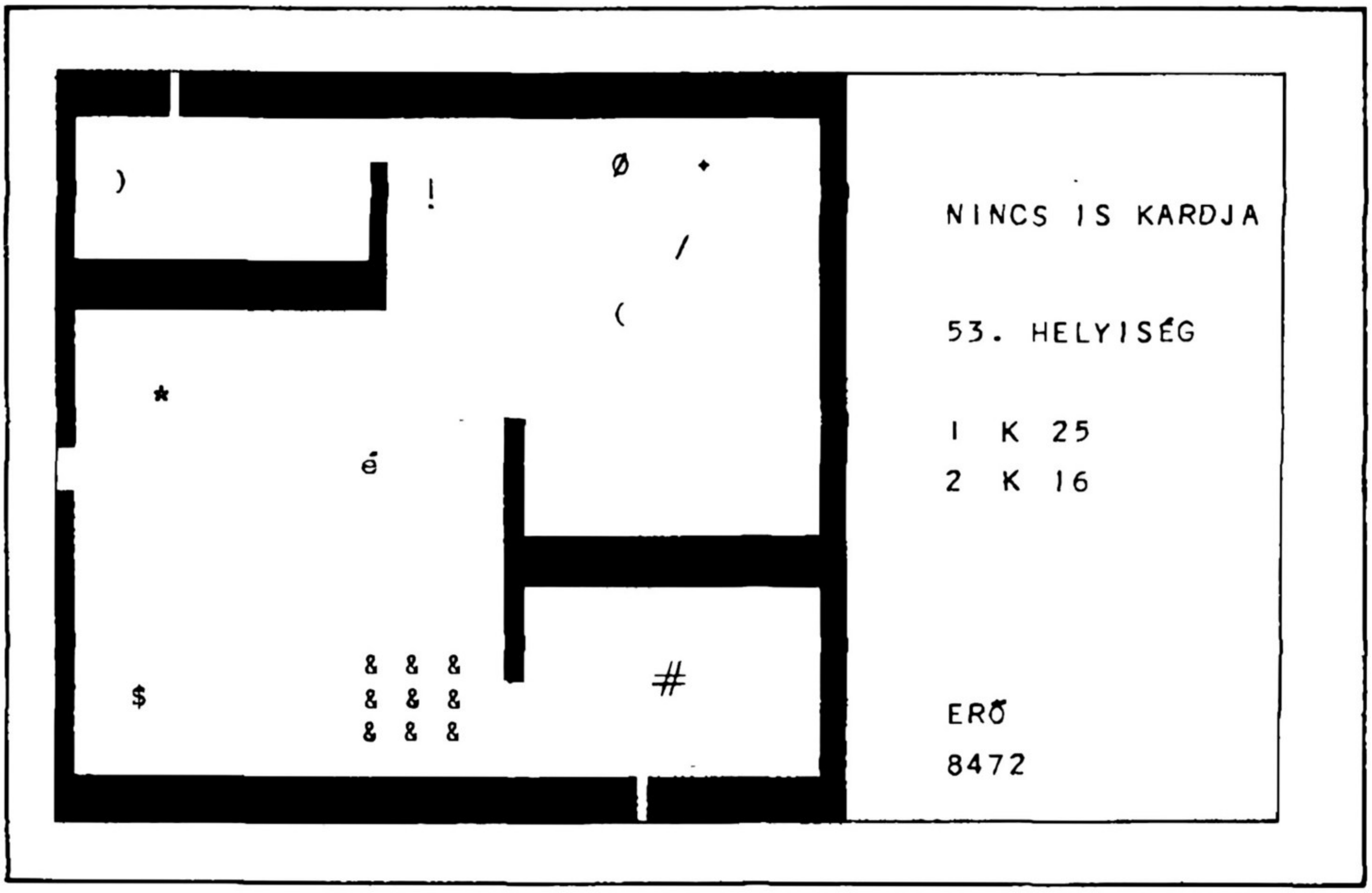
A grafikus kalandok lelke a helyiségek képi megjelenítése. A programozó immár nem érheti be egy hosszú lére eresztett, képzeletet felgyújtó leírás kiírásával; valóban meg kell rajzolnia a helyiséget a képernyőn! Meg kell rajzolni a függőleges, vízszintes és átlós falakat, az átjárókat az előre meghatározott helyeken, ezenkívül valamilyen meghatározott szimbólummal ábrázolni kell az élő és élettelen objektumokat. Akárcsak egy sor más grafikai kalandban, a képernyő a *Szörnyek az útvesztőben* játékban is két részre van felosztva. A bal oldali, nagyobb mezőt akciómezőnek, a jobb oldalit állapotmezőnek nevezzük. Az akciómezőben rajzolódik ki a helyiség képe, és a kalandozó (vagy útvesztőben tévelygő) itt lép kölcsönhatásba a megjelenített dolgokkal. Az állapotmezőben csak szöveg látható, és ott a játék szempontjából lényeges információk jelennek meg. A képrajzolás és az állapotot jelző adatok egy külön területre való kiíratása, miközben nem fedhetik egymást, nos, ez is része lesz az Olvasó újonnan szerzett képzettségének!

Lássuk kissé részletesebben az akciómezőt! A 11-1. ábrán látható egy jellegzetes kép a *Szörnyek az útvesztőben* játékból. A képernyő 88 százalékát az akciómező foglalja el. A mostani helyiség képe grafikus karakterekből álló nyitott négyzetként jelenik meg, amelyet itt-ott a szomszédos helyiségekre nyíló átjárók törnek meg. A kereten belül grafikus blokkokkal ábrázolt falak láthatók (ezeket a helyiség jellegzetességeinek nevezzük). Végül, elszórva látható néhány szabványos karakter, ezek különböző típusú objektumokat jelölnek.

Elsőnek talán a következő kérdés vetődik fel: miért használunk ilyen durva felbontású grafikát? A falakat ugyanis egy 64×16-os rácsban rajzoljuk, akárcsak az alfanumerikus jelek kiíratásakor.

A TRS–80 megenged finomabb felbontású — 128×48 pontból álló — grafikát is, amit a SET, RESET és POINT utasításokkal használhatunk. Akkor vajon nem javítaná-e a nagyobb felbontás az akciómező képi megjelenését?

Azért választottunk mégis durva felbontású grafikát, hogy egyszerűbb legyen a grafikus és alfanumerikus jelek közötti kölcsönhatás. Ha alfanumerikus karakterekkel ábrázolunk egy objektumot, nagyon könnyű a kezelése és különösen a mozgatása. A karakterek azonban szorosan kapcsolódnak a 64×16-os rácshoz. Pl. ha egy átjáró a finom felbontású grafikának megfelelően egy egység, akkor széles, egykarakteres objektumok nem férnek át rajta. A hatékonyság is közrejátszik abban, hogy mindent azonos módon kezelünk a megjelenítő-



11-1. ábra. Egy jellegzetes képernyő a SZÖRNYEK AZ ÚTVESZTŐBEN játékból

mezőben. Mind az objektumokat, mind a falakat POKE paranccsal rajzoljuk és PEEK paranccsal vizsgáljuk. Mindent összevetve sokkal egyszerűbb (és gyorsabb) ha az akciómezőt így rajzoljuk meg, ahelyett, hogy több nagy felbontású grafikus pontból hoznánk létre a tárgyakat, a mozgatásról nem is szólva.

A falakat úgy rajzoljuk meg, hogy a képernyő egymás után következő karakterhelyeit kifehérítjük. Ezt úgy érjük el, hogy a képernyő kívánt helyére POKE-kal beírjuk a 191-es kódú grafikus szimbólumot (a 191-es grafikus karakter épp egy fehér téglalap). A 64×16-os rácsban 1024 lehetséges hely van, ebből 896-ot az akciómező használ föl. Tudván, hogy a képernyő-memória a 15360-as címen kezdődik, minden alakzatot megrajzolhatunk POKE paranccsal.

Bár a külső falon látható átjárók egyszerű szóközöknek tűnnek, valójában nem azok. Az akcióterület nagyrészt a 32-es kódú szóközkarakterrel van feltöltve, az átjárókat azonban a 128-as kódú grafikus karakter jelöli. Ez a különleges karakter üresnek tűnik, mint egy egyszerű szóköz, tehát átjárónak megfelel. A program meg tudja különböztetni egy közönséges szóköztől, ha PEEK utasítással megvizsgálja a képernyő tárát és 128-as értéket talál 32 helyett. Ily módon, ha a játékos egy átjáróhoz érkezik, a program be tudja hívni azt a kezelőt, amely egy új helyiségbe juttatja a programozót.

Minden tárgyat egy alfanumerikus karakter jelképez. A kalandozót egy szabadon mozgatható kereskedelmi egységárjel (@)* szimbolizálja. A lényeket vagy ellenfeleket egy csillag (*) jelképezi, ezek tetszésük szerint támadnak. A kincsekre természetesen adódik a dollárjel ($).

Többféle eszköz is jelen lehet. A fáklya egy felkiáltójel (!), és a játékos csak akkor lát, ha nála van, vagy a helyiségben van. A számjelzés (≠) egy portál, egy olyan titokzatos kapu, ami egy véletlenszerűen kiválasztott helyiségbe juttatja a játékost, ha hozzáér. A pajzs egy kezdő zárójel, amely csökkenti a támadó lények ütötte sebesülés súlyosságát. A befejező zárójel az íj, a kötőjel a nyílvessző. A bűvös ital olyan gyógyszer, amely visszaadja a játékos teljes erejét; a képen pluszjel (+) jelöli. A pont (.) az egyes helyiségekben található, tápláló gyümölcsöt jelképezi. A kard a ferde vonal (/), ez a játékos fő harci eszköze. A kereskedelmi „és" jelekből (&) álló terület tűztengert jelent, ezen a játékos csak nagy fájdalom árán juthat át. Befejezésül, a nulla (0) bombát jelent, bátran lehet szállítani, de elpusztítja azt, aki nem jól kezeli — akár a játékost, akár egy lényt. (A 11-2. ábrán látható a tárgyak teljes listája.)

Bár sok olyan ASCII karakter van, ami még kiírható a TRS–80 modell I és III képernyőjén, de nem használtuk fel. Azért éppen a fentieket választottuk, mert formájuk bizonyos mértékig sugallja azt a tárgyat, amelyet jelképeznek. További karakterek definiálására és használatára is van mód, ha egy hasznos új objektumot sugallnak. Pl. a nagyobb jel (>) kardot jelképezhet, ami egy balul sikerült csatában törött ketté, amikor megpróbált egy páncélos hátú lényt megsebesíteni. Az egyenlőségjel (=) mérgezett dárdás fúvócsövet jelképezhet. A későbbiekben látni fogjuk, hogy a *Szörnyek az útvesztőben* játékot nagyon egyszerű ezen a módon bővíteni.

Az akciómezőtől jobbra látható az állapotmező, ez tájékoztatja a játékost a játszma állásáról. Az állapotmezőbe a Microsoft BASIC PRINT@ utasításával írunk. Ez lehetővé teszi, hogy a szavakat pontosan a megadott helyre írjuk, minden olyan átfedés és kocsivissza-karakter nélkül, amely akaratunk ellenére összezavarná az akciómezőt.

Az állapotmezőt négy ablakra osztjuk fel, ezek mindegyike másféle állapot adatát mutatja. A legfelső az üzenetek ablaka. Itt jelennek meg a beírt parancsokra adott válaszok valamint a játék során szükséges figyelmeztető üzenetek, mint pl. az, amelyik egy veszélyes lény közelségére hívja fel a figyelmet.

A következő a helyiség ablaka, ez mindig megmondja, hogy melyik helyiség látható a képen. A harmadikban, a leltárablakban vannak felsorolva a játékosnál levő tárgyak, 1-től 8-ig önkényesen beszámozva. A játékosnál legföljebb nyolc tárgy lehet. Természetesen a leltár ablaka lehet teljesen üres is, ha nincs a játékosnál semmi. Fel vannak sorolva a külön névvel rendelkező tárgyak; a gyűjtőneveket, mint pl. a 32 kincset, a KI előtag mögé írt sorszám jelzi.

* Az ékezetes HT gépen: É

AZ OBJEKTUM SZÁMA	TIPUS	SZIMBÓLUM	KÓD
1	FÁKLYA	!	33
2		"	34
3	PORTÁL	#	35
4	<KINCSEK>	$	36
5		%	37
6	TÜZ	&	38
7			39
8	PAJZS	(	40
9	IJ	)	41
10	<LÉNYEK>	*	42
11	ITAL	+	43
12			44
13	NYIL	-	45
14	GYÜMÖLCS	.	46
15	KARD	/	47
16	BOMBA	0	48
17-48	KINCSEK	$	36
49-96	LÉNYEK	*	42
49-54	PÓKOK	73-78	ÓRIÁS MÉHEK
55-60	KIGYÓK	79-84	AMŐBÁK
61-66	RÁKOK	85-90	TROLLOK
67-72	SKORPIÓK	91-96	SÁRKÁNYOK

11-2. ábra. A SZÖRNYEK AZ ÚTVESZTŐBEN akadályainak listája

Az utolsó az erő ablaka. Itt mindig látható a játékos harci ereje. Mivel ez a szint másodpercenként változik, az erő ablakát kell a négy ablak közül leggyakrabban frissíteni. Kezdetben a játékos harci ereje 10 000, ezt a mennyiséget azonban fogyasztja az idő múlása, a mozgás, a csatához szükséges erőkifejtés, és leginkább a harcban elszenvedett sérülések. Csak a gyümölcs és a bűvös ital elfogyasztása emeli biztonságos szintre a harci erőt.

Egybillentyűs parancsok

Rég leáldozott már az értelmezést igénylő egy- és kétszavas parancsok napja. A grafikus kalandban minden parancsot egy billentyű leütésével adunk ki. Ez összhangban van az új típusú játék valós idejű jellegével. Elvárható, hogy meggyorsítsuk a parancsok beírását ebben a játékban, amelyik igen nagy mértékben a gyors döntéshozatalra és reagálásra épül. Jól is néznénk ki, ha az egész valós idejű játék sodró lendülete egyszer csak megtörne, amíg a játékos beír egy parancsszót.

A TRS–80 BASIC INKEY$ függvényével a *Szörnyek az útvesztőben* megvizsgálhatja a billentyűk állapotát, miközben áthalad a valós idejű Vezérlőn. Bármelyik billentyű lenyomása, a lenyomott billentyű kódjától függően, közvetlen úton behívhat egy kezelőt.

A 11-3. ábrán láthatók a *Szörnyek az útvesztőben* parancsai. Az objektumokhoz hasonlóan, kívánság szerint bővíthetők a parancsok és a hozzájuk tartozó kezelők; a program felépítése erre módot ad.

BILLENTYŰ	KEZELŐ/PARANCS
NYILAK	MOZGASD A KALANDOZÓT!
V	VÉGY FEL EGY TÁRGYAT!
1-8	TEGYÉL LE EGY TÁRGYAT!
H	KÜZDJ A KARDDAL!
L	LŐDD KI AZ IJAT!
K	FEJEZD BE VAGY PONTOZZ!

11-3. ábra. A SZÖRNYEK AZ ÚTVESZTŐBEN egybillentyűs parancsainak listája

Nyilvánvaló, hogy elsődlegesen a mozgást kell parancsokkal irányítani. A TRS–80 négy nyílbillentyűje igen jól megfelel erre a célra. Átlósan nem mozoghat a játékos, csak két billentyű egymás utáni lenyomásával. Később látni fogjuk, hogy a lényekre nem vonatkozik ez a korlátozás, szükség esetén átlósan is képesek mozogni, hogy elfogják a menekülő játékost.

A nyílbillentyűkkel a játékos vándorolhat, egyesével lépkedhet. Nyilvánvaló, hogy nem hatolhat át falakon, csak az átjárókon. Mozgás közben a program folyamatosan ellenőrzi az utat a játékos előtt. Van akadály az úton? Legtöbbször a játékos útjában álló akadály egyszerűen meggátolja a haladást. A bomba és a portál azonban drasztikus hatással van a mozgásra. A bombával

való találkozás végzetes. Van még egy különleges eset: a tűz. A játékos áthatolhat a tűzön, közben a tűz ki is alszik, de harci ereje jó néhány ponttal csökken.

Ha a játékos egy kapun át próbál távozni, érintkezésbe kerül a 128-as kódú grafikus karakterrel, amely tudatja a programmal, hogy egy bizonyos táblázatból kell adatokat elővenni. A táblázatban a program megtalálja, hogy hová jut a játékos a kapun át, valamint az új helyiség jellegzetességeit, és ennek megfelelően rajzol.

Abban az esetben, ha a fáklya nincs a játékosnál és nincs az új helyiségben, akkor tilos arra menni. Erre külön üzenet figyelmeztet: túl sötét van ahhoz, hogy a helyiségbe beléphessen.

Az egyszerű mozgáson túl, a labirintusban bolyongó játékos szeretne hatni a környezetére. Ahhoz, hogy elsődleges célját — a kincsek felhalmozását — elérje, a játékosnak tudnia kell tárgyakat felvenni, hogy később az 1-es helyiségben, a támaszponton, lerakja őket. A következő két parancs ebből következik.

Ha a játékos a V billentyűt nyomja le, akkor felveszi a közvetlen mellette levő tárgyat. A program végigpásztázza a környezetét — a játékos fölött, tőle balra levő pontból kiindulva — és fölveszi az elsőnek megtalált mozdítható tárgyat. (A lények, a tűz, a falak és hasonlók nem mozdíthatóak.) Ennek megfelelően, ha több mozdítható tárgy is található a közelében, akkor a pásztázási iránynak megfelelően választja ki az elsőt. Igazából csak egy korlátozás áll fenn: a játékosnál egyszerre legföljebb nyolc tárgy lehet. Mint általában a kalandjátékoknál, csökkentené a kihívást, ha a játékos tetszőlegesen sok tárgyat vihetne magával.

A fordítottja is fennáll: a játékos egyesével megszabadulhat a nála levő tárgyaktól. Ezt úgy teheti meg, hogy megnyomja valamelyik számjegy billentyűjét 1 és 8 között. Az állapotmező leltárablakában minden hordozott tárgy mellett egy 1 és 8 közé eső szám áll. Egy tárgyat úgy ejtünk el, hogy az azonosítószámnak megfelelő billentyűt megnyomjuk. Az elejtett tárgyak a játékost körülvevő körbe kerülnek. Ha valamilyen ok miatt nincs hely körülötte az eldobott tárgy számára — pl. egy sarokban van, vagy máris tárgyak veszik körül —, akkor figyelmeztető üzenet jelenik meg az üzenetek ablakán, és több tárgyat nem dobhat el.

A játékos következő tevékenysége a harc. Fő fegyverei a kard és az íj. A kardot a H billentyűvel használja, jelentése Harcolj. Feltéve, hogy a közelben van egy lény, és a kard a játékosnál van, a lény erejének egy bizonyos hányada elvész, a játékos energiájának kisebb mértékű csökkenése mellett. Ha túl messze levő lény felé suhintunk a karddal, vagy a helyiségben nincs is lény, vagy a játékosnak nincs kardja, az üzenetablakban a megfelelő szöveg figyelmeztet rá.

Az L billentyűvel nyílvesszőt lőhetünk egy lényre. (Természetesen a játékosnál kell lenni mind az íjnak, mind pedig a nyílvesszőnek.) Sajnos, a nyílvessző egyszerűen lepattan a túl erős lényekről. Sokszor a nyílvessző célt téveszt, és egy sarokba csapódik be, ahová érte kell menni, hogy újra felhasználhassuk.

Ha azonban megöl egy lényt, akkor rögtön öl, és a játékos ereje nem csökken. A nyílvesszőt ki lehet húzni a lény testéből és újra fel lehet használni.

Befejezésül, a kisegítőparancsok az útvesztő bejárását könnyítik meg. A K billentyű a Kilép parancsot jelenti, hatására a program kiértékeli a jelenlegi állást. A pontszám kiszámítása a megölt lények számán és fajtáján, valamint a felkutatott kincseken alapul, ebből pontokat von le a játékos minden haláláért. (Mert minden halál után feltámad a játékos.) A játékos választhat, hogy akkor és ott befejezi-e a játékot, vagy folytatja, mintha a K billentyűt le sem nyomta volna. Mellékesen a Kilép parancs szinte a teljes állapotmezőt felhasználja. Az állapotmező tartalmát újra felírja a program, ha a játékos a játszma folytatása mellett dönt.

A programozótól függ, hogy ír-e további egybillentyűs parancsokat, amíg a tárban van hely. Az S parancs segítséget nyújthat, pl. a tárgyak és szimbólumaik vagy a parancsok felsorolásával. A P billentyű Pihenést jelenthet, amely távoltartja a halált, mert alvás közben visszatér a játékos ereje.

Átjárók, de hova?

A *Kardhalak és kincsek* játékban a kód jelentős részét a bejárási táblázat kezelésére szántuk. A helyszín, akár szöveges, akár grafikus kalandról legyen szó, nem más, mint a helyiségek halmaza, amit összekötő utak gyanánt az átjárók rendezett listája köt össze. Átjárók nélkül a helyiségek között nem lenne kapcsolat, és nem beszélhetnénk igazi térképes utazási kalandról. Ahogyan a *Szörnyek az útvesztőben* játékban a kapukat kezeljük, az hasonlít is és különbözik is a *Kardhalak és kincsek*-ben használttól. Hasonlítsuk össze a két módszert!

Először is a régi játékban az szabta meg, hogy egy helyiségben hány kapu lehetett, hogy a játékos hány irányban haladhatott. Tíz lehetséges irány van — az iránytű nyolc égtája, és le meg föl —, ezek közül a játékos rendszerint csak keveset használ ki. Emlékezzünk arra is, hogy helyiségen belül nincs elmozdulás; az elmozdulás eredményeként a játékos mindig egy kapun át egy másik helyiségbe kerül.

Az új játékban az akciómező 896 pontjából bármelyik lehet kapu. Ezt az teszi lehetővé, hogy a játékosnak bő mozgási lehetősége van a helyiségen belül. Ezért a nagyfokú szabadságért azt az árat kell fizetni, hogy az új bejárási táblázatban minden kapu pontos koordinátáig meg kell adni. Más szóval, minden kapunak van egy X — vagy vízszintes — és egy Y — vagy függőleges — koordinátája, és a kettő határozza meg a kapu helyét az akciómezőben.

Ezen túl, nem elég annyit tudni, hogy melyik helyiségbe jut a játékos, ha egy bizonyos kaput használ.

Hol a játékos helye, mikor az új helyiséget megrajzoljuk? Az biztos, hogy nem lehet csak úgy véletlenszerűen a középre tenni! Bármilyen is legyen az új

bejárási táblázat, tartalmaznia kell mind a helyiségnek a számát, ahova a játékos jut, mind az (X, Y) koordinátákat, amelyeken megjelenik. Ezek a koordináták az új helyiség megfelelő kapujába helyezik a játékost. A kapuk koordinátáit és a helyiséget, ahova a játékos kerül, szisztematikusan és logikusan kell megtervezni.

A *Szörnyek az útvesztőben* játékban a bejárási táblázat helyett minden helyiséghez egy karakterlánc tartozik. A helyiségekhez tartozó láncok kódolt formában tartalmazzák a kapuk koordinátáit, és hogy a kapun áthaladva milyen koordinátákra jutunk. A helyiségek láncai tartalmazzák a helyiség megrajzolásához szükséges adatokat is.

A cél elérése

Végső soron a *Szörnyek az útvesztőben* játéknak hármas célja van: (1) az életben maradás, (2) az összes kincs begyűjtése, (3) az összes lény megölése.

Sok minden akadályozza eszerint a célokat. Pl. a játékos még akkor is éhen halhat, ha nincs ellenség a közelben, mert nem talál élelmet. Kincs bőven van, de ezek el vannak zárva a színhely 90 helyiségében — és 90 helyiség az rengeteg! Nem könnyű a lényeket megölni, viszont azok „betájolják" a játékost és megtámadják. Nem, ebben a játékban nem könnyű sok pontot szerezni! Szerencsére, a játékba bele van szőve néhány segítség. Az első, hogy a játék a bázison, az 1-es helyiségben kezdődik, és két hasznos eszközt odakészítettünk a játékos számára: a fáklyát és a kardot. Ezenkívül: az 1-es, 2-es, 3-as helyiségbe véletlenül sem téved be lény, ha veszélyes helyzetbe kerül a játékos, mindig idemenekülhet.

Mi a helyzet az éhséggel és a kimerültséggel? A játékos ereje pihenés közben fogy, mozgás közben még gyorsabban fogy. Két táplálék áll rendelkezésére. Bizonyos helyiségekben egy darab bűvös gyümölcs van, ez újraterem, amikor a helyiséget elhagyjuk. Aki elfogyasztja, annak az ereje részben visszatér. A játékos tartsa észben ezeket a helyiségeket, ha rájuk akadt, és igyekezzen valamelyiknek a közelében maradni!

A gyümölcsön kívül van egy varázsital is, egyfajta gyógyszer. Ez időnként csak úgy előttünk terem egy helyiségben. Aki elfogyasztja, annak a teljes 10 000 pontnyi ereje visszatér. A varázsital eltűnik, hogy ismét felbukkanjon egy másik helyiségben. Előfordul, hogy a játékos ráakad a varázsitalra, de nem használja fel, ilyenkor jól jegyezze meg, hogy melyik helyiségben van.

Hogy nehogy a játékos nagyon elkényelmesedjen, jó ha megjegyzi a következő intelmet: A lények is szeretik az élelmet és az orvosságot! Bizony ám! Ha egy lény abban a helyiségban van, ahol az élelmiszer-utánpótlás, épp úgy megeheti, mint mi magunk! Ugyanez áll a varázsitalra is, ami aztán eltűnik, hogy egy másik helyiségben bukkanjon fel. Lehet, hogy ez lehangoló, de ki állította, hogy a kalandprogram fenékig tejföl?

A kincsek felkutatása viszonylag egyszerű, ha betartjuk a szokásos korlátozásokat. Egyszerre legföljebb nyolc tárgyat vihetünk magunkkal, és ez valójában hat kincset jelent egy út alkalmával, mivel szükség van a fáklyára, hogy lássunk, és csak a bolond hagyja el a Kardot. A kincseket a bázison, az 1-es helyiségben kell lerakni, hogy a pontozásba beszámítsanak.

Egy lényt megölni már nem olyan egyszerű! Megkönnyíti a dolgunkat, hogy egyszerre legföljebb egy lény lehet egy helyiségben. Az az egy is bőven elég lesz! A szörnyek mérete a halálos csípésű póktól a hatalmas sárkányig terjed, és a nagyobb lények ereje kezdetben nagyobb, mint a játékosé. Ingerlés nélkül is újra meg újra támadnak. Átlósan is tudnak mozogni, míg a játékos csak vízszintesen és függőlegesen mozoghat.

Fegyverek nélkül halálra vagyunk ítélve. A kardunk az erőnkkel arányos méretű sebet ejt az ellenségen. A vadnak nagyon közel kell lennie, hogy a karddal kárt tehessünk benne. Az íj és a nyíl annál pontosabb, minél közelebb van a lény. Még akkor sem biztos, hogy a nyíl eltalálja, ha egészen közel van a lény. Igazából a nyíl levágódik az erősebb lényekről, míg az erejük kissé ki nem merül.

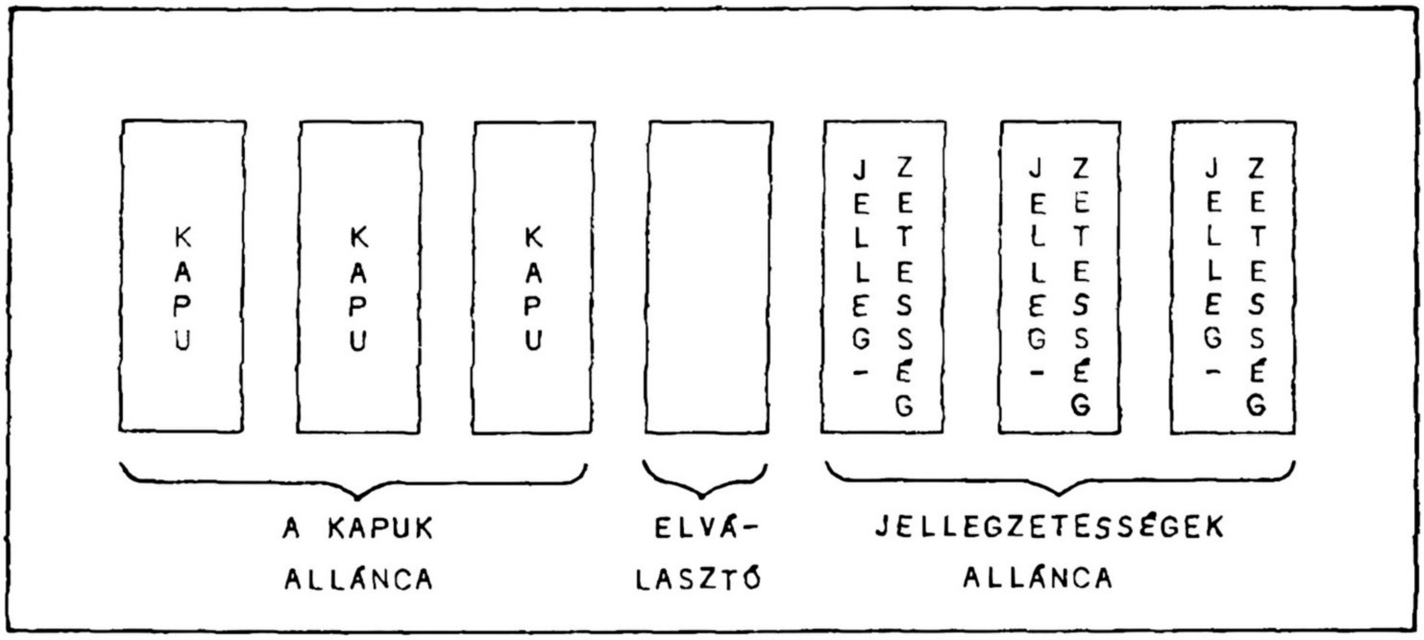
Van némi védelem is. Először is ott a pajzs, amely kissé csökkenti a lények által okozott sérülést. A bombát a lény útjába lehet dobni. Ha az ellenség rálép, darabokra tépi — de *minket* is, ha ügyetlenek vagyunk és rálépünk! Ha minden kötél szakad és fel kell venni a nyúlcipőt, akkor ott a portál. Ezt a földre kell dobni, bele kell lépni, és villámgyorsan egy másik — remélhetően biztonságos — helyiségbe röpíti a játékost.

A helyiségek láncai

A *Kardhalak és kincsek*-ben minden helyiséghez kétfajta információ tartozott. Először is a helyiségek leírásának blokkja — ebben minden helyiséghez bizonyos szöveg tartozik —, ez egyben a helyiség részletes leírása. (Ugyanebben a blokkban van a helyiségek rövid neve is, amit a második esettől alkalmazunk.) Másodszor, a bejárási táblázat mondja meg, hogy hova jutunk, ha egy bizonyos helyiségben egy megadott irányba mozdulunk.

A *Szörnyek az útvesztőben* játékban másfajta adatok veszik át a fentiek helyét. A bejárási adatok helyett kapura vonatkozó információk vannak, a szöveget pedig a helyiség kirajzolásához szükséges adatok váltják fel. Mindkét információt az ún. helyiségláncban tároljuk.

A 11-4. ábrán látható a helyiséglánc alapvető szerkezete. Minden helyiséghez tartozik egy lánc, amely két alláncra bomlik. Az első a kapualláncok halmaza, ez a helyiségben található kapuk helyzetét és működését tartalmazó számkódokból áll. A második a jellegzetességek halmaza. Ebben a falak és a többi jellegzetesség — pl. tűzfolt vagy bűvös gyümölcs — megrajzolásához szükséges számkódok vannak. A két alláncot kötőjel választja el. Az elválasztó



11-4. ábra. Egy helyiséglánc összetevői

mindig jelen van, még abban az elméleti esetben is, ha nem lenne jellegzetességeket tartalmazó allánc.

Nézzük először a kapuk alláncát! Ügyeljünk arra, hogy a grafikus kalandjátékban minden kapuról három adatot kell feljegyezni! Ezek a kapu helyzetét jelző X és Y koordináták a képernyőn, annak a helyiségnek a száma, ahová a kapu vezet, és annak a pontnak a koordinátái (X és Y), ahova a játékost rajzolni kell, mikor az új helyiségbe érkezik.

Mennyi helyet foglalnak el ezek az adatok? Az X és Y koordináták közül X 0 és 55 közé, Y pedig 0 és 15 közé esik. Az első tételnél 4 számjegy kell X és Y együttes ábrázolásához, a harmadik tétel négy további számjegye nyolcra növeli a szükséges számjegyek számát. A második tételben a helyiség száma 1 és 99 közötti egész; így a kapuallánc hossza összesen tíz számjegy.

A *11-5.* ábrán látható a kapuk alláncának felosztása. Mivel minden kapuhoz pontosan tíz számjegy hosszúságú allánc tartozik, nincs szükség elválasztókarakterre. Az alláncokat kezelő szubrutin tudni fogja, hogy tíz karakter többszöröseivel kell lépkednie, hogy eljusson egyik alláncról a másikra. Ha ez a bizonyos szubrutin számjegy karaktert talál, akkor tudja, hogy van még kapuallánc; ha kötőjel elválasztót talál, akkor pedig nincs több kapu a helyiségben.

A falak és a megkülönböztető jellemzők szerepe nemcsak abban áll, hogy segítségükkel könnyebb egy helyiséget felismerni, a játékot is nehezítik azzal, hogy akadályozzák a mozgást. Ezen túl, egyes különlegességeknek, mint pl. a bűvös gyümölcsnek a helyzetét is meg kell adni.

Hatféle dologgal lehet egy helyiséget díszíteni a *Szörnyek az útvesztőben* játékban. Az első négy különféle fal, mégpedig vízszintes, függőleges, valamint felfelé és lefelé dűlő átlós. Az ötödik jellegzetesség a tűzmező, ezt szükség esetén egy kapu lezárására is felhasználhatjuk. A hatodik a bűvös gyümölcs.

KARAKTEREK	ADAT
1 - 2	A KAPU X KOORDINÁTÁJA
3 - 4	A KAPU Y KOORDINÁTÁJA
5 - 6	CÉLHELYISÉG
7 - 8	UJ X KOORDINÁTA
9 - 10	UJ Y KOORDINÁTA

11-5. ábra. A kapualláncok összetevöi

A hat jellemző megrajzolásához különböző jellegű adatokra van szükség. Az első négy jellemző egyenes vonal. Ezeknél az értelmező-szubrutinnak meg kell kapnia a vonal típusát, a kiinduló pont X, Y koordinátáit és a vonal hosszát. Az 5. és 6. jellemző esetén a hossz megadására nincs szükség, változatlanul kell viszont a típus és az X, Y koordináták. Tehát a jellegzetességeknél kétfajta allánc van — egy hosszabb és egy rövidebb. A hosszabb hét karakter hosszú, a rövidebb csak öt.

A 11-6. ábrán látható a jellemzők alláncának ötkarakteres és hétkarakteres felosztása. A rajzoló-szubrutin az allánc elején álló típusjelző számból meg

KARAKTEREK	ADAT
1	A JELLEGZETESSÉG TIPUSA
2 - 3	A KEZDŐPONT X KOORDINÁTÁJA
4 - 5	A KEZDŐPONT Y KOORDINÁTÁJA
(6 - 7)	(A VONAL HOSSZA)
TIPUS	**JELLEGZETESSÉG**
1	VIZSZINTES VONAL (-)
2	EMELKEDŐ LEJTŐ (/)
3	FÜGGŐLEGES VONAL (|)
4	SÜLLYEDŐ LEJTŐ (\ )
5	TÜZ (9 & JELBŐL ÁLL)
6	GYÜMÖLCS (EGYETLEN .)

11-6. abra. A jellegzetessegek allancanak összetevoi

tudja állapítani az allánc hosszát, így elválasztás nélkül is megtalálja a következő allánc elejét.

A beírás és a visszakeresés megkönnyítése érdekében a 90 helyiséghez tartozó alláncok külön sorba kerültek, és külön tárterületet foglalnak el. A hozzáférés megkönnyítése érdekében, az alláncok egy 90 elemű karakterláncvektor elemeihez vannak hozzárendelve. A karakterláncvektor használata nem jár a láncok kétszeri tárolásával, csak egy mutatókból álló lista jön létre, ez a program területén elhelyezett karakterláncokra mutat. Ehhez mintegy 200 bytenyi hely kell a karakterláncvektor-változómutató számára. Az eredmény azonban kárpótol ezért — az akciómezőt a gyors hozzáférés következtében igen gyorsan fel tudjuk frissíteni.

A 90 helyiséges térkép

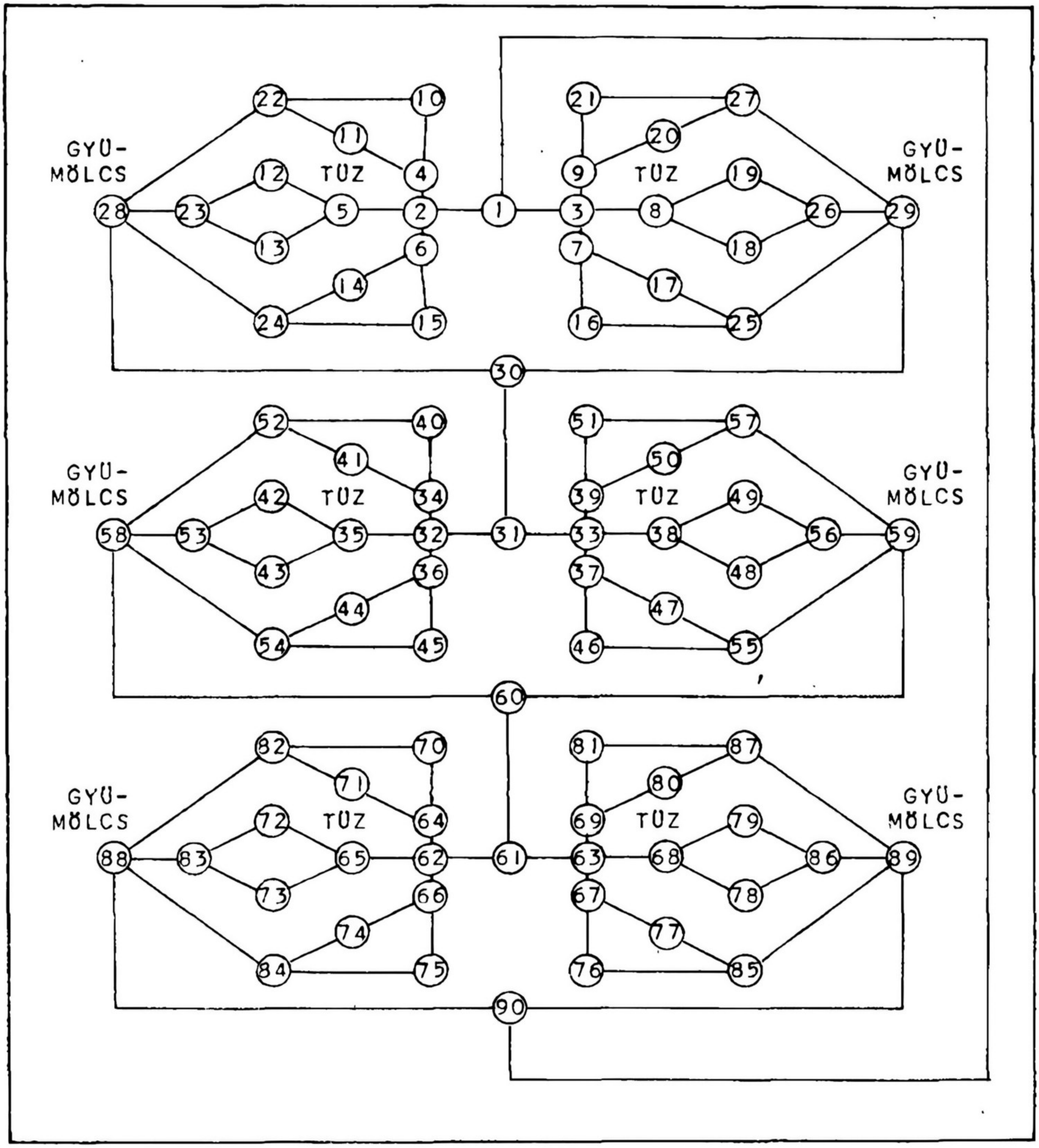
Mint ahogy a *Kardhalak és kincsek*-ben szükség volt a helyiségek közötti kapcsolatot ábrázoló térkép megrajzolására, a grafikus kalandjáték helyszínének térképét is meg kell rajzolni. Itt azonban nincs értelme az összekötő utakat É, D, K címkékkel jelölni. Az égtájaknak itt nincs jelentősége. Most csak kapuk vannak az akciómező megfelelő képernyőpontjaiban.

Ebben az esetben értelmetlen megrajzolni a helyszín részletes térképét. Változatlanul hasznos azonban a helyiségeket feltérképezni az összekötő utakkal együtt, tekintet nélkül a kapuk pontos helyére, azért, hogy látni lehessen, melyik helyiségbe vezetnek. Később, szükség szerint, meg lehet határozni a kapuk pontos helyét úgy, hogy beleilleszkedjenek a durva vázlatba.

A 11-7. ábrán látható a *Szörnyek az útvesztőben* játék térképe. Figyeljük meg, hogy az összekötő vonalak helyzete nagyjából megfelel a kapuk végső helyzetének! Bejelöltük, melyik helyiségben található valamilyen különlegesség — tűzfolt vagy gyümölcshalom —, hogy majd bevegyük a helyiség láncába.

A tervezés leegyszerűsítése érdekében választottuk ezt a szimmetrikus alakú térképet. Látni fogjuk, hogy a helyiségek alakja az akciómezőben szintén követi ezt a szimmetriát. Ez természetesen a helyiség láncán alapuló önkényes dolog. Ideális esetben minden egyes helyiséget teljesen eltérőre kellene megtervezni.

Itt is érvényesülnek azok a szabályok, amelyek a szöveg típusú kalandjátékok színhelyének térképezésénél. Figyeljük meg, hogy néhány elágazás is szerepel! Ezeknek az a célja, hogy csökkentsük annak a lehetőségét, hogy egy ág mentén az egészet be lehessen járni. A játékos a helyiségek új rétegére talál, valahányszor egy másik kapun át hagy el egy csomópontot, úgy maximálisra növeljük a játékos bizonytalanságát.

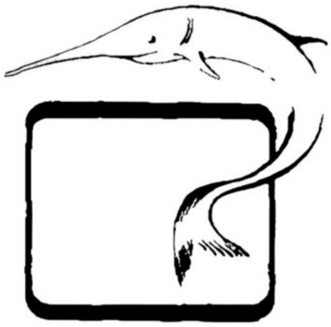


11-7. ábra. A SZÖRNYEK AZ ÚTVESZTŐBEN teljes helyszíntérképe

Lássuk a konkrét részleteket!

Most már ismerjük azokat az általános jellegzetességeket, amelyek egy *Szörnyek az útvesztőben* jellegű grafikus kalandot megkülönböztetnek a szöveges programoktól. Ezt a programot sok tekintetben könnyebb elkészíteni. Az biztos, hogy a kód rövidebb. Minden egyszerűségével együtt a grafika vonzó játékká teszi.

Most már készen áll az Olvasó arra, hogy rátérjünk a *Szörnyek az útvesztőben* részleteire, a változók kijelölésétől kezdve a kezelők szerkezetén és működésén át egészen a szubrutinokig.



12. FEJEZET

Grafikus kalandok: a szegmensek

Kezdjük a program szerkezetével! A szöveges kalandokhoz hasonlóan itt is elsőrendű cél, hogy a kód könnyen áttekinthető, bővíthető vagy módosítható legyen. Lényegénél fogva a BASIC nem éppen ilyen nyelv; tehát a programozó dolga, hogy a szerkezetet létrehozza.

A 12-1. ábrán látható a program szerkezete. Figyeljük meg, mennyire hasonlít a *Kardhalak és kincsek* szerkezetére! A *Szörnyek az útvesztőben* sokkal egyszerűbb.

A program első szegmense a 0-s és 99-es sor között található inicializáló rész — természetesen ezekből a sorszámokból csak néhányat használ fel. Ebben a részben deklaráljuk az összes tömböt és kezdőértéket adunk a változóknak. Figyeljük meg, hogy az objektumok — például a kincsek és lények — helyét véletlenszerűen jelöljük ki ebben a részben. Ez természetesen különbözik attól, ahogy a szöveg típusú kalandnál jártunk el, hiszen ott szigorúan egy kitöltött táblázatból vettük a kezdőértékeket.

Ezt követi a Vezérlőhurok a 100 és 199 közötti sorokban. Ezt a részt azért nevezzük huroknak, mert a program a futási idő nagy részében a kódnak ezen a részén fut körbe-körbe. A program még akkor is újra meg újra átfut ezen a szegmensen, ha a játékos nem ad ki parancsot. Ilyenkor a játékos erejét módosítja, egy lényt mozgat — ha éppen van a közelben —, egy szóval fenntartja a dolgok állapotát. A *Szörnyek az útvesztőben* azzal érdemli meg a valós idejű játék elnevezést, hogy ez a hurok állandóan működik.

A *Kardhalak és kincsek*-hez hasonlóan, a játékos parancsait itt is a kiválasztott kezelő hajtja végre. Ez a kezelő kifejezetten arra a célra íródott, hogy a kívánt hatást létrehozza.

A kezelők a program 200-as és 499-es sora között helyezkednek el. Általában minden parancsra jut egy kezelő, a kezelőket pedig egy megadott billentyű lenyomása hívja be. A megjelenítőrész egy szükséges kiegészítő rész, amely az 500-as és 599-es sor között található. Ezt akkor hívja meg a program, amikor a kalandozó belép egy új helyiségbe. Megrajzolja a szoba falát és a

SORSZÁMOK	PROGRAMSZEGMENS
0-99	INICIALIZÁLÁS
100-199	VEZÉRLŐHUROK
200-499	KEZELŐK
500-599	MEGJELENITŐRÉSZ
900-999	SZUBRUTINOK
1000-1099	A HELYISÉGEK LÁNCAI

12-1. ábra. A SZÖRNYEK AZ ÚTVESZTŐBEN programszerkezete

többi jellegzetességet, és egyben jelzi a helyiségben található objektumokat. A szubrutin nagymértékben támaszkodik a helyiségek láncaiban tárolt információra. Ezek feltöltése később történik.

Következik a szubrutinrész: 900 és 999 között. A szubrutinokat vagy a Vezérlőhurok, vagy a megjelenítő-szubrutin, vagy az egyes kezelők hívják meg.

A legutolsó (és legnagyobb) programszegmens a helyiségláncoké. Ez az 1000-es sortól fölfelé található. Minden helyiség lánca külön sorban van, így az 1-es helyiség lánca az 1000-es, a 90-es helyiségé az 1089-es sorban. Minden egyes sorban a szöveg típusú vektor egyik elemébe belekerül a helyiség lánca. A helyiségláncok szegmense tulajdonképpen az inicializáló rész által behívott egyetlen nagy szubrutin, és mindössze a helyiségláncok kiolvasásához szükséges változómutatóknak ad kezdőértéket. Ennek megfelelően a szegmenst RETURN utasítás zárja le.

Minden, amit a változókról tudni kell

Karbantartás szempontjából a *Szörnyek az útvesztőben* játék a szöveges kalandoknál egyszerűbb. Adminisztrációs célra kevesebb változót igényel. A 12-1. ábrán felsoroltuk a változókat.

A helyszínen 96 objektum van: 16 eszköz, 32 kincs és 48 lény. Minden egyes objektumról fel kell jegyezni, melyik helyiségben van. A helyiség számát az L(*n*) vektor elemeiben tároljuk, ahol *n* az objektum száma 1 és 96 között. (Az objektumok listáját l. a 11-2. ábrán.) Ehhez hasonlóan a játékos is egy bizonyos helyiségben tartózkodik, aminek a számát L(0)-ban tároljuk.

Az L(0) és L(96) közötti változók egy-egy helyiség számát adják meg, értékük 1 és 90 között lehet. Ezenkívül, amikor egy lényt megölnek, akkor a hozzá tartozó L(*n*) értéke 0 lesz, mert a 0-s helyiség a sír. A 91-es számot rendeljük azokhoz a tárgyakhoz, amelyeket a játékos magával visz a hátizsákjában.

A program működéséhez szükség van még jó néhány értékre, de némi

tárterületet nyerünk azzal, ha ezeket az L(n) vektor további elemeiben tároljuk. Mi ennek az oka? Az ok az, hogy a programban használt minden egyes változóhoz tartozik egy változómutató is. Megtakaríthatunk néhány byte-ot akkor, ha megpróbáljuk a lehető legtöbbet kihozni egy már létrehozott változóból.

Az élettelen tárgyak helyét lényegében véletlenszerűen jelöljük ki az akciómezőben — amint ezt később meglátjuk. Az előbbi állítás nem igaz a játékos és az ellenséges lény esetében — ha van ilyen a helyiségben —, mivel mindkettőnek szabadon és értelmesen kell mozognia. Ezért folyamatosan feljegyezzük helyzetüket a helyiségben. A helyzetet X, Y alakban tároljuk, ahol X a vízszintes koordináta karakteregységekben kifejezve (értéke 0 és 55 közé esik), Y a függőleges koordináta karakteregységekben (értéke 0 és 15 közé esik). L(97)-ben és L(98)-ban tároljuk a kalandozó X és Y koordinátáját, L(101)-ben és L(102)-ben pedig a lény helyzetét. Nyilvánvaló, hogy L(101) és L(102) tartalma érdektelen, és elhanyagolható abban az esetben, ha a helyiségben nincs lény.

Még szükség van néhány tényezőre, ami a játékosra vonatkozik. A játékos ereje 10000-ről indul. Folyamatos hatást gyakorol rá a harc, a kimerültség vagy a táplálkozás. Az L(99)-es elem őrzi a játékos erejét. Lehet, hogy a játékosnak nem sikerül megölnie az útvesztő undorító szörnyeit, és egy sötét helyiségben meghal. Mivel a meghalások száma beleszámít a játékos végső pontszámába, L(100)-ban tartjuk számon a halálesetek számát.

Az L(n) vektor hátralevő elemeinek többsége az esetlegesen a helyiségben tartózkodó lényhez kapcsolódik. L(103) mutatja meg a lény erejét, ami éppen olyan széles határok között változik, mint a játékosé. A lény számát L(104)-ben tároljuk. Ha a helyiségben nincs lény, akkor L(104) értéke 0, ez tudatja a szubrutinokkal, hogy hagyják figyelmen kívül az összes lénnyel kapcsolatos változót.

A helyiségben tartózkodó lény kétféle módon viselkedhet: vagy támad, vagy visszavonul. Támadás közben egyenesen a játékos felé tart; visszavonulás közben egyenes vonalban távolodik tőle. Az L(105)-ös változó 1 és –1 között változik, ezzel jelezve a támadást vagy visszavonulást. A lény mindaddig támad, amíg el nem éri a kalandozót, vagy a kalandozó el nem hárítja karddal vagy nyíllal. Ezután véletlenszerűen megállapított számú lépésen át hátrál. A visszavonulást L(106)-ban állítjuk be és csökkentjük egyesével. Mikor a számláló nullához ér, a lény ismét támadásba lendül.

(Talán észrevette az Olvasó, hogy minden változó azt tételezi fel, hogy egy helyiségben legföljebb egy lény van. Ez így is van rendjén, mivel az inicializáló rész úgy van megírva, hogy a 48 lényből legföljebb egy kerülhet egy helyiségbe. A 32 kincs tetszőleges módon elosztható a helyiségek között.)

Az utolsó, az L(107)-es elemet időzítésre használjuk. A Vezérlőhurok elég gyakran ismétlődik; túl gyors ahhoz, hogy ilyen gyakorisággal csökkentsük a lény és a játékos erejét. A hurok úgy készült, hogy csak minden tizedik lefutásakor csökkentse egy egységgel az ellenfelek erejét. L(107) egy tízes osztású

VÁLTOZÓ	ALKALMAZÁSA
L 0 -L 96	AZ OBJEKTUM HELYE
L 97	A KALANDOZÓ X KOORDINÁTÁJA
L 98	A KALANDOZÓ Y KOORDINÁTÁJA
L 99	A KALANDOZÓ EREJE
L 100	A HALÁLOK SZÁMA
L 101	A LÉNY X KOORDINÁTÁJA
L 102	A LÉNY Y KOORDINÁTÁJA
L 103	A LÉNY EREJE
L 104	A LÉNY SZÁMA
L 105	TÁMADÁS/VISSZAVONULÁS-JELZŐ
L 106	VISSZAVONULÁS-SZÁMLÁLÓ
L 107	SZÁMLÁLÓ
R 1 -R 90	A HELYISÉGEK LÁNCAI
C 1 -C 8	A HÁTIZSÁK TARTALMA
TA$	A LELTÁRI TÁRGYAK LÁNCA
TB$	A LÉNYEK NEVEINEK LÁNCA
CRS	A JELENLEGI HELYISÉG LÁNCA

12-2. ábra. A SZÖRNYEK AZ ÚTVESZTŐBEN változói

számlálóként működik, valahányszor nullára csökken az értéke, mindig ismét 10-et töltünk bele.

A következő indexes változó valójában szöveg típusú: R(*n*). A típust jelző $ karaktert elhagytuk, mivel az inicializáló részben DEFSTR utasítással szöveg típusúra deklaráltuk R(*n*)-t. Ezzel soronként egy byte-ot takarítunk meg a 90 soros helyiséglánc részben. Ebben a részben az R(1)-től R(90)-ig terjedő tömbelemeknek értéket adunk a programsorba írt helyiségláncokkal. Ez az elrendezés nem igényel külön területet a tár magas címein szövegek számára fenntartott részből. Ehelyett a változókba mutatók kerülnek, amelyek a program területén tárolt helyiségláncokra mutatnak. Ily módon egy bizonyos helyiség jellegzetességeit R(*n*) megfelelő elemére hivatkozva egyszerűen megtalálhatjuk, a módszer gyors, a kompromisszum ésszerű.

A játékos legföljebb nyolc tárgyat vihet a hátizsákjában. Már tudjuk, hogy a hordozott tárgyakhoz a 91-es helyiségszám tartozik az L(*n*) vektorban. A hátizsák kezelését nagyon leegyszerűsíti, ha a tartalmát mindig nyilvántartjuk. A C(*n*) vektor látja el ezt a feladatot. A C(1) és C(8) közötti tömbelemek mindegyike valamelyik hordozott tárgy számát tartalmazza. A tárgyak felvételét és

lerakását kezelő szubrutinok biztosítják, hogy a listában ne legyen hézag akkor sem, ha nyolcnál kevesebb tárgyat hordoz a játékos. Más szóval, a felvett tárgyat C(*n*) első szabad elemében jegyezzük fel, a fel nem használt elemeket nullára állítjuk. Pl. ha négy tárgy van a játékosnál, akkor ezek száma C(1)... ...C(4)-ben van. A C(5)...C(8) elemek értéke nulla. Az állapotmező leltár nevű ablakát frissítő szubrutin C(*n*) segítségével sorolja fel a hordozott tárgyakat.

Amikor a leltárablak felfrissítésére kerül sor, a C(*n*)-ben tárolt tárgysorszámokat a játékos számára érthető tárgynevekké kell átalakítani. Ebből a célból létrehozunk egy TA$ nevű karakterláncot a tárgyak nevének tárolására. A láncban 16 darab hatbetűs név van, egy-egy az 1 és 16 közötti sorszámú tárgyak — az eszközök — számára. (Jelenleg a definiálatlan tárgyak helyén a formális „ABCDEF" név áll.) A megfelelő szubrutin némi számolás és szövegkezelés árán ebből a 96 karakteres láncból ki tudja választani a keresett hatkarakteres nevet. A szintén hordozható kincseknek nincs neve, ezeket egészen más módon kezeljük a leltárablakban.

Hasonlóan, a különböző típusú lényeknek is van azonosítónevük. A 48 lényt nyolc csoportra osztjuk: a gyenge póktól a rettenetes sárkányig. Amikor valamelyik lénnyel összetalálkozunk egy helyiségben, akkor az üzenetablakot kezelő szubrutinnak ki kell olvasnia a nyolc lény nevét tároló karakterláncot. Ebből a célból létrehozunk egy TB$ nevű karakterláncot: ez tárolja a lények nevét. A láncban nyolc darab nyolckarakteres név található, a nyolc karakternél rövidebb neveket szóközökkel egészítjük ki. A helyes nevet itt is elővehetjük egy kis számolás és szövegkezelés árán.

Az utolsó névvel jelölt változó CR$, ez a mostani helyiség láncát tárolja. A CR$ változó valójában inkább kényelmi célokat szolgál. Valahányszor egy helyiségbe belépünk, CR$-ba kerül a jelenlegi helyiség lánca R(*n*)-ből. Az ezt követő szubrutinok egyszerűen CR$-ra hivatkozhatnak, nem kell egy tömbelemhez fordulniuk. Egyébként a megfelelő helyiségláncra csak a kényelmetlen R(L(0)) kifejezéssel hivatkozhatnánk.

Miután befejeztük a játék változóinak áttekintését, fordítsuk figyelmünket a program kódja felé! A leírás nagy részében a *Szörnyek az útvesztőben* 13. fejezetben közölt listájára hivatkozunk.

Inicializálás

A program inicializáló része a következő feladatokat hajtja végre:

- köszönti a játékost,
- deklarálja az indexes változók méretét és a karakterláncoknak helyet foglal le,
- véletlenszerűen elhelyezi a kincseket,

- véletlenszerűen elhelyezi a lényeket,
- definiálja a karakterláncokat,
- véletlenszerűen vagy előre kijelölt helyre elhelyezi az eszközöket,
- kezdőértéket ad a játékos változóinak.

Az inicializálás a 2-es BASIC sorban kezdődik. A program kiírja a játék nevét, eléje írja a 23-as kódú különleges karaktert, ezzel ideiglenesen 32 karakteres kiírási módon állít be, hogy a felirat tetszetősebb legyen. Az inicializáló rész végén álló CLS utasítás hatására visszaáll a közönséges kiírási mód.

Ezután helyet biztosítunk a szövegek számára, és deklaráljuk az indexes változókat. A CLEAR 256 utasítás alapállapotba hozza a változókat, és lefoglal 256 byte-ot a karakterláncok részére. Ez bőven elég az esetleges karakterlánc-műveletekhez és az aktuális helyiség láncának kezeléséhez. (A játékban használt karakterláncok zömét explicit módon megadjuk, és a program területén tároljuk.) A DEFINT utasítással az összes változót egész típusának deklaráljuk. Ezzel tárolóterületet és időt takarítunk meg.

A legfontosabb szám jellegű vektorokat — L(*n*)-t és C(*n*)-t — DIM utasítással deklaráljuk. Ezután a DEFSTR utasítással az R betűvel kezdődő változókat szöveg típusúnak deklaráljuk. Ennek az a célja, hogy helyet takarítsunk meg, amikor az R(*n*) vektor 90 elemének értéket adunk. A következő utasításban DIM utasítással deklaráljuk a vektort. Majd meghívjuk az 1000-es sorban kezdődő szubrutint, amely az R(*n*) vektor megfelelő elemeibe tölti a helyiségek láncait.

Nézzük csak az 1000-es sorban kezdődő helyiségláncok szegmensét! Figyeljük meg az egyes sorok szerkezetét: mindegyik egy-egy karakterláncot definiál! Némi türelemmel és az előző függelék felhasználásával az Olvasó biztosan elemezni tudja mind a kapuk, mind a jellegzetességek alláncát, ebből pedig némi fogalmat alkothat arról, hogyan néznek ki az egyes helyiségek. A későbbiekben részletesen tanulmányozzuk azt a programrészletet, amelyik valóban elvégzi az elemzést.

Az inicializálás 4-es sora a négy égtáj felé szétszórja a 17 és 48 közötti tárgyakat — a kincseket. Az egyes tárgyakhoz tartozó L(*n*) tömbelembe 4 és 90 közötti véletlenszámok kerülnek. Tehát bármelyik helyiségben találhatunk kincset, kivéve a bázist (1-es helyiség), és a vele közvetlenül szomszédos két helyiséget (2-es és 3-as helyiség). Akármelyik helyiségben lehet egynél több kincs is.

A lények elhelyezése, ami a 6-os és a 8-as sor között történik, eltér ettől. A lények kezelésének módjából következik, hogy egy helyiségben legföljebb egy lény lehet. Ebből a célból létrehozzuk az X$ változót, ez helyiségenként egy karaktert tartalmaz. A VARPTR függvény segítségével kiszámítjuk az X$-ben tárolt első karakter tárbeli címét és ezt P-be töltjük. Ezután minden lénynek (49 és 96 közti objektumok) véletlenszerűen kijelölünk egy helyiséget.

A módszer lényege, hogy X$-ban feljegyezzük, melyik helyiségben van már lény, ezeket a számokat nem fogadjuk el, ha a FOR-NEXT ciklusban

ismét felbukkannak. A véletlenszerűen kiválasztott helyiségnek megfelelő karaktert megvizsgáljuk, hogy szóköz-e (32 a kódja), és ezért szabad, vagy kötőjel-e (45 a kódja), azaz már foglalt. Ha szóköz, akkor a lényt a helyiségbe tesszük, és a szóközt kötőjelre változtatjuk, hogy a helyiséget ne lehessen újra kiválasztani. Ha nem szóköz, akkor a 8-as sor önmagára tér vissza, és egészen addig állít elő véletlenszámokat, amíg szabad helyiséget nem talál. A helyzetet jelölő L(*n*) tömbelembe a kiválasztott helyiség száma kerül. Miután az összes helyiség kiválasztása megtörtént, az X$ munkaváltozóba a nulla hosszúságú szöveget töltjük, ezzel lényegében kivonjuk a forgalomból.*

Ismét figyeljük meg az RND(87)+3 kifejezést! Ez 4 és 90 közé eső véletlenszámokat állít elő. A kincsekhez hasonlóan lények sincsenek a helyszínnek abban a bizonyos első három helyiségében.

A 10-es sorban létrehozzuk a karakterláncokat: TA$-t, a tárgyak nevének láncát, és TB$-t, a lények nevének láncát. Ügyeljünk rá, hogy a szóközöknek fontos szerepe van mindkét láncban, mert a keresett nevet úgy választjuk ki, hogy karakterenként leszámoljuk a láncokat!

A 14-es sorban bizonyos eszközöket véletlenszerűen elhelyezünk a 4 és 90 közötti helyiségek egyikében. Figyeljük meg, hogy eltérően a 4-es sortól, itt nem használunk ciklust, mivel nem minden eszközt helyezünk el véletlenszerűen, és az 1 és 16 közötti tárgyak között vannak definiálatlanok is! (Ha egy definiálatlan tárgy kerül egy helyiségbe, akkor az akciómezőben ezt egy karakter jelzi — pl. a 12-es tárgynál egy vessző. A tárgyat fel lehet venni és el lehet vinni, de semmi értelme. A leltárban a tárgy neve „ABCDEF”.)

Ezután a 16-os sorban egyes kiválasztott tárgyakat az 1-es helyiségben helyezzük el. Az 1-es fáklya és a 15-ös kard alapvetően szükséges ahhoz, hogy az útvesztőben haladjunk, ezért ezeket segítőkészen a bázison helyezzük el. A későbbiekben látni fogjuk, hogy ezek akkor is visszakerülnek a bázisra, mikor a játékos meghal és fel kell éleszteni. Legvégül beállítja a játékos erejét és helyzetét. L(0)-ba 1 kerül, így a játékos a bázison van. Az X és Y koordináta kb. az akciómező közepére teszi a játékost. Erejét a 10000-es kezdőértékre állítja be.

Egy CLS utasítás fejezi be az inicializálást: törli a képernyőt — eközben visszaállítja 64 karakteres módba is —, ezzel előkészíti az első helyiség kirajzolását. Mindent együttvéve a játék előkészítése összesen öt másodperc késleltetést okoz.

*Tapasztalatunk szerint ez az utasítás felesleges. (*A fordító*)

Megjelenítés

Bár az inicializálást a vezérlőciklus kódja követi, a ciklus egyik legelső dolga, hogy kirajzolja az akciómezőbe a jelenlegi helyiséget. A helyiséglánc kiolvasása és a helyiség megrajzolása a megjelenítőrész feladata. A megjelenítőrész az 500-as sorban kezdődik. A Dsplay néven emlegetett szubrutint tartalmazza.

Dsplay a következő lépéseket látja el az akciómező kitöltése és a játékosnak az új helyiségbe történő beléptetése céljából:

- alapállapotba hozza a lényeges változókat,
- egy üres keretet rajzol az akciómezőbe,
- megrajzolja a kapukat,
- megrajzolja a jellegzetességeket,
- megrajzolja a játékost,
- megrajzolja az összes objektumot,
- ha van lény a helyiségben, akkor beállítja a hozzá kapcsolódó változókat.

Dsplay azzal kezdi, hogy nullát ír L(104)-be. Ez a változó mutatja meg, hogy melyik lény van jelen. Ha a szubrutin 7. lépése másként be nem állítja, a nulla azt jelzi, hogy a helyiségben nincs lény. Ezután L(0) segítségével, amely a jelenlegi helyiség számát tartalmazza, kiolvassuk az R(n) vektor megfelelő elemét, és betöltjük a helyiségláncot CR$-ba.

A következő lépésben kirajzoljuk az akciómező szélét alkotó négyszögletes keretet. Ezt úgy lehet leggyorsabban végrehajtani, hogy karakterláncokat hozunk létre, és kiíratjuk őket a képernyő megfelelő helyére. (Ez jóval gyorsabb módszer, mintha egy ciklussal végigjárnánk a képernyő megfelelő pontjait, és egyesével POKE-olnánk a karaktereket.) Van még egy előnye ennek az eljárásnak: mindent kitörlünk az akciómezőből, ami esetleg korábban ott volt, anélkül, hogy CLS utasítással az egész képernyőt letörölnénk. Így a képernyő állapotmező részét nem zavarja, hogy a játékos egyik helyiségből a másikba vándorol.

Először a keret alját és tetejét rajzoljuk meg, úgy, hogy 191-es kódú fehér téglalap karakterekből létrehozunk két 56 karakteres láncot és kinyomtatjuk. Ezután a belsejét szóközökkel töltjük fel, és megrajzoljuk a keret oldalait úgy, hogy többször kinyomtatunk egy láncot, amely csupa szóközből áll, az elején és a végén egy-egy fehér téglalappal.

Most már a szubrutin készen áll, hogy átnézze a helyiségláncot és kirajzolja a kapukat a képernyőre. A láncot a MID$ függvénnyel vizsgáljuk, az N változó mutat az egymást követő karakterekre. A ciklus előkészítéseként az 500-as sor végén N értékét 1-re állítjuk be.

Az 502-es sor az N változó által kiválasztott karakter vizsgálatával kezdődik. Ha ez egy kötőjel, akkor a szubrutin tudja, hogy elérte a kapuk alláncának végét, és rátérhet a feladat következő részére. Egyébként elkezdi elemezni a soron következő néhány karaktert a 11. fejezetben ismertetett elvek szerint.

Emlékezzünk arra, hogy minden kapu allánca tíz karakterből áll: az első kettő a kapu kétjegyű X koordinátája, a következő kettő a kétjegyű Y koordináta! A MID$ függvénnyel kiválasztjuk ezeket a karakterpárokat, és a VAL függvénnyel előállítjuk az eredeti számértékeket. Amikor a koordináták az X és Y változóba kerültek, POKE utasítással a képernyőre rajzoljuk a kaput. Az X+64*Y+15360 kifejezés az X,Y koordinátapárból a képernyőtár egy megadott rekeszének címét állítja elő. Megjegyzem, hogy a kaput a 128-as kódú karakterrel ábrázoljuk; annak ellenére, hogy ez úgy néz ki, mint egy szóköz, valójában egy üres grafikus karakter, és ezt a különbséget a program képes érzékelni.

Miután egy bizonyos kapu alláncot kiolvastunk, 10-et adunk az N változóhoz — ezzel az alláncot átlépjük. Ezután ismét az 502-es sort hajtjuk végre. Az 502-es sor végrehajtása addig ismétlődik, míg kötőjelet nem találunk. Ezért van szükség minden helyiség alláncában kötőjelre, még akkor is, ha a jellegzetességek allánca el is marad. Kötőjel hiányában az 502-es sor tovább olvasna a CR$ változó végénél, ami hiba volna.

Amikor az összes kapu kirajzolása befejeződött, az 504-es sor elkezdi a jellegzetességek alláncainak kiolvasását. Az N változó máris egy karakterrel a kötőjel mögé mutat, készen a következő pásztázásra. Az 504-es sor először azt vizsgálja meg, hogy elértük-e már a CR$ végét. Ha nem, akkor a soron következő karaktereket megvizsgáljuk a jellegzetességek alláncának már korábban ismertetett szerkezetének megfelelően.

Mindenfajta jellegzetességnél a másodiktól a negyedik karakterig a jellegzetesség kezdetének X,Y koordinátája található. A koordinátákat — az 502-es sorhoz hasonlóan — MID$ és a VAL függvénnyel választjuk ki. Ezután kell kiválasztani a jellegzetesség típusát. Az allánc első karaktere 1 és 6 közötti számjegy. Leválasztjuk a számjegyet, és egy ON GOTO utasítás segítségével a jellegzetességhez tartozó sorra adjuk a vezérlést. A különböző jellegzetességeket megrajzoló sorok 506 és 512 között vannak.

Az 1-es típusú a vízszintes vonal. Az 506-os sor nagyon gyorsan kirajzolja a vízszintes vonalakat, mert a BASIC PRINT@ utasítását használja, ahelyett, hogy POKE-kal egyesével rajzolná meg a vonal pontjait. Az X+64*Y kifejezés adja meg a megfelelő képernyőcímet a PRINT@ utasításnak. A 191-es kódú grafikus karakterekből egy láncot hozunk létre. A hosszát a vonal alláncának hatodik és hetedik karakteréből állapítjuk meg; a számértéket ismét előállítjuk, és a STRINGS$ függvény ezzel működésre kész. Miután a láncot kiírtuk, N-et héttel megnöveljük, hogy a következő jellegzetesség elejére mutasson. Visszatérünk az 504-es sorra, és megvizsgáljuk, hogy van-e még kódolt jellegzetesség.

Az 508-as sor rajzolja meg az összes többi vonalfajtát. Az S változó követi nyomon a vonal egyes pontjainak kirajzolását. A pontok POKE utasítással kerülnek a képernyőre. A már kiválasztott X és Y koordináta segítségével S-et beállítjuk a vonal kezdőpontjának megfelelő képernyőcímre.

Ha az Olvasó belegondol, rájön, hogy nagyon egyszerű kitalálni, milyen címre kerül a vonal következő pontja. Nézzük először a függőleges vonalat! Minden pont éppen 64-gyel magasabb címen van az előzőnél. Ugyanis a TRS–80 gépen egy sor 64 karakter hosszú. Tehát függőleges vonalat úgy rajzolunk, hogy S-t 64-esével növeljük.

Mi történik, ha S-t csak 63-mal növeljük? A következő pont az előző alatt, attól balra lesz. Ezt ismételgetve végeredményben egy emelkedő átlós vonalat kapunk. Hasonlóan, ha S-t 65-ösével növeljük, süllyedő, átlós vonalat kapunk.

A vonal típusát ebben a három esetben azért választottuk meg pont így, mert a vonal típusához 61-et hozzáadva, éppen megkapjuk azt, amennyivel S-et kell növelni. Az 508-as sor leválasztja a típust jelző számot, átalakítja, hozzáad 61-et, az eredményt pedig D-ben tárolja. Ezután lefuttatunk egy ciklust egytől a vonal hosszáig. A vonal hosszát most is az allánc 6. és 7. karakteréből kapjuk meg. D tartalmát használjuk arra, hogy a vonal a megfelelő szögben hajoljon, és a 191-es kódú grafikus karaktert POKE utasításokkal sorozatosan beírjuk a képernyő tárcímeire. A folyamat végén megkeressük a következő alláncot.

Az 5-ös jellegzetesség egy tűzfolt. Ezt kilenc kereskedelmi „és" jellel (&) jelezzük. A mezőt a már kiválasztott X és Y koordinátájú pont mint középpont köré rajzoljuk. A gyorsaság érdekében ismét PRINT utasítást használunk POKE helyett. A mezőt úgy hozzuk létre, hogy kinyomtatunk háromszor három & jelet. Az első kezdőcíme az X,Y koordinátájú pont felett, attól egy hellyel balra van. A PRINT utasítás kezdőpozícióját úgy számítjuk ki, hogy a középpont kezdőcíméből 65-öt kivonunk.

Az 510-es sor rajzolta a tűzfoltot. Egy sorba kiírunk három & jelet, a kezdőcímet minden sor kiírása után 64-gyel növeljük, és így tökéletes háromszor hármas négyzetet kapunk. Ezután az allánc mutatóját, N-et, öttel növeljük, hogy a következő alláncot olvashassuk ki.

Az utolsó jellegzetesség a bűvös gyümölcs, képe egy pont (.), amit az X és Y által meghatározott pontra kell rajzolni. Az 512-es sor kiszámítja a pont címét a képernyőtárban, és egy POKE utasítással lerakja a pont ASCII kódját, a 46-ot. Az alláncot átlépjük, és újra végrehajtjuk az 504-es sort.

Miután befejeztük a különböző jellegzetességek megrajzolását, meg kell rajzolni a játékost. Az L(97) és L(98) vektorelemekben van a játékos X és Y koordinátája a helyiségen belül: a játékost mozgató szubrutin alapállapotba hozza ezt a két változót. Ha szabályos kapun át lépünk a helyiségbe, akkor a játékost közvetlenül a kaputól jobbra kell megrajzolni. A POKE utasítással egy 64-es kódú karaktert teszünk a megfelelő helyre, ez a képernyőn egy kereskedelmi egységárjelet (@) eredményez — a játékos szimbólumát.

Ezután a helyiségben található objektumok megrajzolása következik. A rajzolás három szakaszban történik, mivel háromféle objektum van.

Először az eszközöket rajzoljuk meg, ezek száma 1 és 16 közé esik. Egy

közönséges ciklussal ellenőrizzük az egyes eszközök helyzetét. Kirajzoljuk az összes tárgyat, amelyeknek helyzete megegyezik a jelenlegi helyiség számával — ez az L(0) változóban van. Az eszközt ábrázoló karaktert úgy kapjuk meg, hogy a tárgy sorszámához 32-t hozzáadunk. Így éppen a korábban leírt szimbólumokat kapjuk.

Hova kell az eszközöket rajzolni? Az eszköz X,Y koordinátáit nem találjuk meg egyik változóban sem. Ezért véletlenszerűen helyezzük el őket a képernyőn. Meghívunk egy olyan szubrutint, amelynek az a feladata, hogy egy elfogadható X,Y koordinátapárt állítson elő. A szubrutin neve Randxy, és a 940-es sorban található. Véletlenszámokat állít elő: X 1 és 54 közé esik, Y 1 és 14 közé. Ez az akciómező keretén belül véletlenszerűen jelöl ki egy pontot. Az eszközt azonban nem rajzolhatjuk rá egy már kirajzolt jellegzetességre! A Randxy PEEK-kel megvizsgálja a kiválasztott pontot. Ha üres karaktert talál, akkor visszatér, egyébként addig választ ki újabb pontokat, amíg egy megfelelőt nem talál.

Ugyanezt az eljárást követjük a 17 és 48 közötti kincsek esetében. Randxy segítségével helyet keresünk a kincseknek a helyiségben. Ezúttal azonban a 36-os számot tesszük a képernyőtárba, ez a dollárjel ($) kódja.

Legvégül ellenőrizzük, hogy 49 és 96 közötti objektum — lény — van-e a helyiségben. Ebben az esetben megszakítjuk a FOR-NEXT ciklust, ha egyezést találunk, mert egy helyiségben legföljebb egy lény lehet. A Randxy segítségével helyet keresünk a lénynek, és a megfelelő címre POKE-kal elhelyezünk egy 42-es kódú csillagkaraktert.

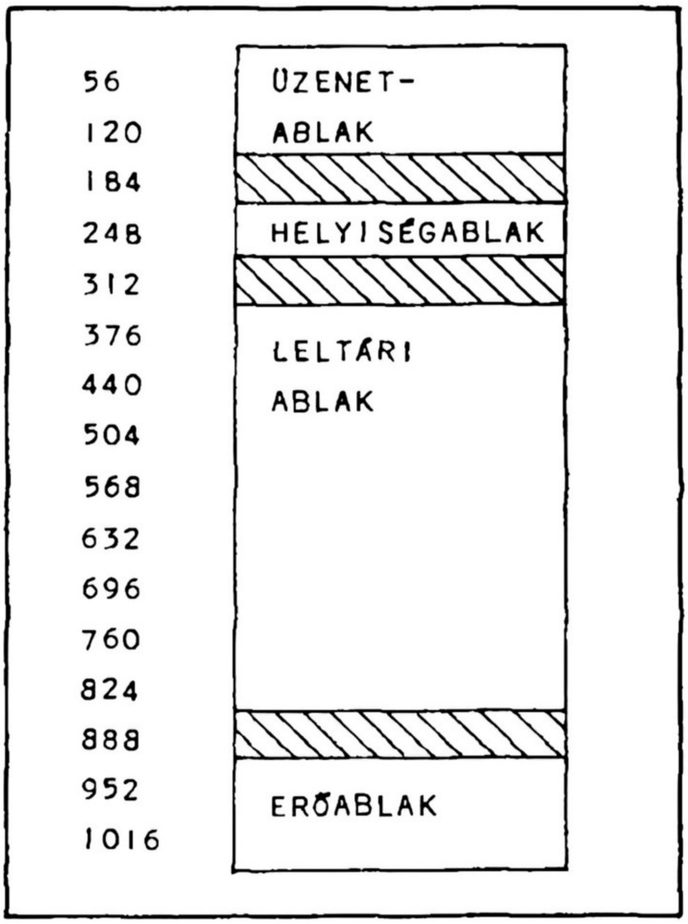
Feltéve, hogy a helyiségben nincs lény, Dsplay ezzel befejeződik, és visszatér a hívó szubrutinba. Abban az esetben, ha van lény, be kell még állítani néhány változót. Az L(101) és L(102) tárolja a lény X és Y koordinátáját, amiket a Randxy-tól kapott értékre állítunk be. Az L(104)-be (amelynek értéke nulla, ha nincs lény) a lény objektumszámát tesszük. Ezután a Dsplay visszatér.

A Vezérlőciklus

A Vezérlőciklus kódja 100-tól a 130-as sorig terjed. A körülményektől függően a Vezérlőhurok a következő feladatokat látja el:

- frissíti az üzenetablakot és az akciómezőt,
- inicializálja a lényt (ha van),
- frissíti az állapotmezőt,
- felülírja az erőablakot,
- kezeli a bejövő parancsot,
- irányítja a lény mozgását vagy támadását,
- kezeli a játékos halálát és feltámasztását.

Az első feladat egyszerű. Meghívjuk a 920-as sorban található Clrmes szubrutint, hogy törölje az üzenetablakot (Clrmes egyszerűen üres karakterek-



12-3. ábra. Az állapotmezőhöz tartozó képernyőcímek

ből álló láncokat ír az üzenetablak két sorába az 56-os és 120-as címtől kezdődően). Ezután a Dsplay szubrutin újraírja az akciómezőt. A következő lépésben megvizsgáljuk az L(104) változót, hogy van-e lény a helyiségben. Ha nulla az értéke, akkor nincs lény, és áttérünk a harmadik lépésre. Egyébként kiszámítjuk a lény erejét és L(103)-ban tároljuk. Az erő attól függ, hogy a nyolc közül milyen fajta lénnyel van dolgunk. A lény erejének kiszámításánál felhasználjuk a lény L(104)-ben tárolt számát. A lény ereje az egyszerű pók 5000 pontjától a félelmetes sárkány 12050 pontjáig terjed.

Az üzenetablakban megjelenik a lény neve a „VIGYÁZZON" figyelmeztetés kíséretében. A lény 8 karakteres nevét a TB$ karakterláncból olvassuk ki a *MID$* függvény és a lény számának segítségével. Az így kapott nevet kiírjuk. Végül az (L(105)-öt 1-re állítjuk be. Ez a változó azt jelzi, hogy a lény támad-e vagy visszavonul: 1 azt jelenti, hogy támad, −1 jelzi a visszavonulást.

Akár van lény, akár nincs, a következő lépés az állapotmező felfrissítése. Ezt a 900–902 sorban található Status szubrutin meghívásával intézzük el. A 12-3. ábrán látható az állapotmező elrendezése. Feltüntettük a PRINT@ utasításban használt képernyőcímeket is. A Status szubrutin először a fejlécet írja ki, valamint a helységablakhoz és az erőablakhoz tartozó értékeket. Ezután egy ciklussal megvizsgáljuk *C(n)*-t — a hordozott tárgyak vektorát. *C(n)* azon elemeinél, ahol egy tárgy számát találjuk — tehát a tárgy a játékosnál van — egy 1 és 8 közötti sorszámot írunk ki. Ha a tárgy kincs, akkor a KI jelzést íratjuk

ki, ezt követi a kincs 1 és 32 közé eső száma — ez egyszerűen a tárgy számánál 16-tal kevesebb. Ha a tárgy eszköz, akkor TA$-ból kiolvassuk a tárgy hatkarakteres nevét, és kiírjuk. A ciklust rögtön megszakítjuk, ha C(*n*)-ben nulla értékű elemet találunk. Az utolsónak kiírt tárgy sora után még egy üres sort is kiírunk, hogy eltüntessük az előző leltár maradékát. Ezután az L(99) változóból kiírjuk az erőablakba a játékos erejét.

Az INKEY$ függvény segítségével megvizsgáljuk, hogy írtak-e parancsot a billentyűkön. Háromféle parancs van: (1) a nyíllal jelölt billentyűk a Move kezelőt hozzák működésbe, (2) a számjegyek billentyűi a Drop kezelőt hívják be, (3) a betűbillentyűk a parancsokat azonosítják.

Ha egy nyíl jelű billentyűt ütünk le, akkor az INKEY$ függvény a 8-as, 9-es, 10-es, ill. 91-es kódú ASCII karaktert adja eredményül. A Vezérlő elvégzi ezt a vizsgálatot, és a 260-as sorra, a Move kezelőre adja a vezérlést, ha ez az eset fordul elő. Egyébként meghívja a Clrmes szubrutint, hogy az üzenetablakot előkészítse a következő parancsoknak. (Könnyebb megállapítani, hogy mikor van kiírva új üzenet, ha két parancs között töröljük az üzenetablakot.)

Ha valamelyik számjegybillentyűt nyomták le, akkor a Vezérlőhurok átadja a vezérlést a 230-as sorban kezdődő Drop kezelőnek. Amikor olyan karakter érkezik, amelynek kódja kívül van a betűk ASCII kódtartományán, a Vezérlő figyelmen kívül hagyja a parancsot.

A 106-os sorban találhatók az egykarakteres parancsok kezelőinek BASIC sorszámai. Az angol ábécé minden betűjéhez tartozik egy sorszám: összesen tehát 26 sorszám van. Amelyik betűhöz nem tartozik parancs, annál a megfelelő sorszám a parancsfeldolgozás mögé mutat, így a betű figyelmen kívül marad. Maguk a kezelők — dolguk végeztével — a Vezérlő néhány belépési pontjának valamelyikére térnek vissza.

A lény mozgatása

Ha feltételezzük, hogy nem írtak le parancsot, akkor a lény mozgatása következik. A 110-es sor megvizsgálja L(104)-et: vajon van-e lény a helyiségben. Ha nincs, akkor a következő lépést kihagyja. Egyébként előállítjuk a lény javasolt következő lépését. Ez úgy történik, hogy összehasonlítjuk a játékos és a lény jelenlegi X és Y koordinátáit. A lény új helyzete a játékoshoz egy fokkal közelebb vagy tőle egy fokkal távolabb lesz. Az L(105)-ös változót — amely a támadást vagy visszavonulást jelzi, értéke 1 vagy −1 — szorzóként használjuk a mozgási irány meghatározásakor.

Mielőtt a lépést megtennénk, megvizsgáljuk a javasolt lépés következményeit. A vizsgálatot úgy végezzük, hogy megnézzük az új helyen a képernyő tartalmát. Ez ASCII kód formájában a D változóban van. Hat esetet különböztetünk meg:

- érintkezés a játékossal,
- érintkezés a bombával.
- érintkezés a gyümölccsel,
- érintkezés az itallal,
- szabad hely,
- akadály.

Ha a játékossal érintkezik, akkor a 112-es sor figyelmeztető üzenetet ír ki. Ez a lény nevéből és a „TÁMAD” szóból áll.

(Mint korábban, most is a lények nevét tartalmazó láncból választjuk ki a megfelelő nevet.) Ezután villogtatni kezdjük a képernyő megfelelő pontját, hogy így jelezzük: a játékost megharapták. A villogást úgy érjük el, hogy gyorsan váltogatjuk a 191-es kódú fehér téglalapot és a játékos 64-es kódú karakterét a játékos helyén. A lény hátrálni kezd — ezért −1 értéket töltünk L(105)-be —, vagyis a játékostól távolodik. A visszavonulás számlálójába 1 és 20 közötti véletlen értéket töltünk: lehet, hogy a lény messze menekül, de lehet, hogy szinte azonnal újra támad.

Kiszámítjuk, hogy mekkora sérülést okozott a támadás a játékosnak. A játékos erejét a lény erejének egynyolcadával csökkentjük. Tehát az erősebb lény súlyosabb sebet ejt. Ha van a játékosnak pajzsa, ez kissé védi. Megvizsgáljuk a 8-as tárgyat, a pajzsot, hogy benne van-e a játékos zsákjában (91-es helyiség). Ha igen, akkor a játékos erejét 500 ponttal növeljük a támadás után. A 114-es sorban kiértékeljük a támadás következményeit: meghalt a játékos? Ha nem, akkor végrehajtjuk a Vezérlőciklus következő lépését. Ha igen, akkor végrehajtjuk a 128-as sort, amely a játékos halálát és feltámadását kezeli.

A játékos halálakor lefuttatunk egy ciklust, amelyik kiüríti a C(n) vektort, és a hordozott tárgyak abba a helyiségbe kerülnek, ahol a játékos meghalt. Ezután eggyel megnöveljük az L(100) változót, amely a pontozás miatt számolja, hogy hányszor halt meg a játékos. Töröljük a képernyőt, és egy üzenetet írunk ki. A 128-as sorban az INKEY$ ciklikusan vizsgálja, hogy lenyomták-e a NEW LINE billentyűt. Ez jelzi, hogy a játékos készen áll a folytatásra. A program úgy folytatódik, hogy visszatér a 16-os sorra. Így a játékos visszakerül a bázisra (1-es helyiség), mellékerül a fáklya és a kard — mindez így tisztességes, hiszen nélkülük nem jutna messzire.

A fentiekre akkor kerül sor, ha egy lény a játékossal találkozik. Mi történik, ha a bombához ér? A 116-os sor lép működésbe. Ha hozzáért, egy ciklus véletlenszerűen grafikus karaktereket kezd villogtatni a bomba helyén — a bomba felrobban. A pont kialszik, és a bomba — a 16-os tárgy — véletlenszerűen átkerül a 3-as és 90-es közötti helyiségek egyikébe.

A 124-es sor intézi a lény halálát. Az akciómezőből eltűnik a lény képe: a POKE-kal egy szóközt írunk a helyébe. Az üzenetablakban a „VÉGRE KIMÚLT” feliatot kapjuk. A lény a nullás helyiségbe kerül — itt már csak a pontozószubrutin találja meg —, majd az L(104)-beli nulla jelzi a Vezérlőciklusnak, hogy a mostani helyiségben nincs már lény. A program a 126-os

sorban folytatódik, hogy módosítsa a játékos erejét — hamarosan látni fogjuk, hogyan.

Mi történik, ha a lény a gyümölcshöz vagy italhoz jut? A 117-es sor foglalkozik ezzel az esettel. A gyümölcs esetében a gyümölcsöt jelképező karakter helyére POKE-kal szóközt írunk, a lény ereje pedig egy véletlen értékkel megnő. Ugyanígy az ital esetén is letöröljük a szimbólumot. A lény ereje ilyenkor 10000-re növekszik, az ital pedig — a 11-es tárgy — a 3-as és 90-es közötti helyiségek valamelyikébe kerül.

Mi történik, ha a lény szabad helyre lép? Szabad az út: a 118-as sor törli a lény mostani helyét, L(101)-be és L(102)-be beírja a lény új X és Y koordinátáját, és POKE utasítással megrajzolja a lény szimbólumát az új helyre. A 120-as sorban — ha L(105)-ben −1 van — csökkentjük a visszavonulás számlálóját. Ha a visszavonulás számlálója nullára csökken, az L(105)-ben tárolt támadás/visszavonulás jelzőt 1-re állítjuk át, és a lény támadásba lendül. A 122-es sorban a lény erejét módosítjuk, ha az L(107)-ben tárolt számláló túlcsordul. (Ez a számláló tízzel leosztja a Vezérlőciklus ismétlődését.) Ha a lény ereje elfogy, akkor sorra kerül a halálát kezelő 124-es sor.

Végül mi történik, ha a lény akadályba ütközik: pl. falba vagy egy eddig nem tárgyalt objektumba? A 119-es sor ilyenkor támadásra indítja a lényt, ha éppen visszavonulóban van. A lények új erőre kapnak az akadálytól és elszánttá válnak, mintha falig hátráltak volna. Ha a lény már támad, akkor mintha nem látná tisztán, hogyan is férhet a játékoshoz, más utat kell választania. Ezt úgy éri el, hogy véletlenszerűen kiválaszt legföljebb három új X,Y koordinátapárt. Ha a három közül az egyik szabad helyre mutat, akkor a program a 118-as sorra ugrik, és közönséges esetként veszi. Egyébként úgy tűnik, mintha a lény elakadna, mert meg kell várnia a vezérlőciklus következő lefutását, hogy kiszabaduljon.

Ami a ciklusból még hátravan

A 126-os sor teszi teljessé a Vezérlőhurkot, itt módosul a kalandozó ereje. A módosítás, épp úgy, mint a lénynél, a ciklus minden tizedik lefutásakor történik meg. Eggyel csökkentjük az L(107)-es számlálót: ha nem csordul túl, akkor elölről kezdődik a ciklus a 104-es sortól, vagyis a parancs elemzésétől. Ha túlcsordul, akkor ismét tizet töltünk bele. Ezután egy ponttal csökkentjük a kalandozó erejét. Ha a csökkentés eredményeként az erő nullára csökken, a kalandozó meghal, és a program a 128-as sorban, a játékos halálával folytatódik.

A Move kezelő

A négy, nyíllal jelölt billentyű hívja be a 260-as és a 280-as sor között álló Move kezelőt. A Move-nak meg kell határoznia, hova akar a játékos lépni, és fel kell készülnie arra, hogy különböző dolgokba ütközik.

Először Move a lenyomott billentyű alapján meghatározza az eredő mozgásirányt. A négy billentyű közül a „fel" nyíl rí ki, mivel a 91-es kód tartozik hozzá. A lenyomott billentyű kódja még mindig a D változóban van, ahova a Vezérlőciklus tette. A 260-as sor úgy módosítja D értékét, hogy szabályosabb legyen, így a négy nyíl a 8, 9, 10 és 11 értéket adja. Ezeknek a számoknak a felhasználásával a 262-es sor olyan sorokra adja a vezérlést, amelyek a nyilak által kijelölt irányokat X és Y értékekké alakítják át. A pozitív irányú elmozdulásnak 1 felel meg, a negatív irányúnak −1; a helyben maradásnak pedig 0. Ezután az előállított számokat hozzáadjuk a játékos mostani X,Y koordinátáihoz; ezt nevezzük a játékos javasolt helyzetének. PEEK-kel megvizsgáljuk, hogy mi van a képernyőn a játékos javasolt pozíciójában, és az eredményt Q-ban tároljuk.

Ha a játékos szabad helyre lép, akkor a 274-es sor letörli a játékos régi helyét, L(97)-be és L(98)-ba teszi a módosított X,Y koordinátákat, az akciómezőben megrajzolja a játékost az új helyén, és öt ponttal csökkenti a játékos erejét — érzékeltetve az elfáradást. Ha az elfáradás azt jelenti, hogy a játékos ereje nullává vagy negatívvá válik, akkor a játékos halálát kezelő 128-as sorszámú szubrutin következik, egyébként a Vezérlőciklus elölről kezdődik.

Amikor a játékos a Bombához ér, a 276-os sor a következő üzenetet írja ki az üzenőablakban: „BUMM! BOLOND". Egy ciklusutasítás hatására a képernyőn a bomba véletlenszerű grafikus karaktereket kezd villogtatni. Majd a bomba egy másik helyiségbe kerül, és a játékos halálát kezelő szubrutin végrehajtása következik.

A portálhoz érve, a 278-as sor lép működésbe, előállítva egy helyiségszámot 3 és 90 között. Ha a fáklya nincs abban a helyiségben és a játékosnál sincs — ezt a fáklya helyét tároló L(1) vizsgálatával állapítjuk meg —, akkor megjelenik a „SEMMI SEM TÖRTÉNIK!" üzenet, és ismét a Vezérlőhurok kerül sorra. Különben az akciómezőben, a játékos közelében megjelenik, hogy „PUFF!". Némi késleltetés után a játékos helyét jelző L(0) változóba az új helyiség száma kerül, és a Vezérlőbe egy olyan korai ponton lépünk be, amely az akciómezőt is újra megrajzolja.

Ha tűzhöz ér a játékos, a 280-as sor törli a régi helyén, kirajzolja az újon — nagyon hasonlóan ahhoz, mintha szabályosan szabad helyre lépne. Eközben azonban az ereje egy véletlenszerűen előállított mennyiséggel csökken, és megjelenik a „TŰZ!! JAJ!!" üzenet. Szokás szerint, ha elfogy a játékos ereje, sorra kerül a halált kezelő szubrutin.

Kapuknál (128-as karakter) a 282-es sor megvizsgálja: van-e a játékosnál fáklya. Ha nincs, akkor a 288-as sor megakadályozza az elmozdulást, és figyel-

meztető üzenetet ír ki: „*TÚL SÖTÉT VAN OTT BENN.*" Különben a 282-es sorban egy ciklus összehasonlítja a kapu koordinátáit a jelenlegi helyiség láncának kapualláncával. Egyezés esetén a játékos helyét tároló L(0) változóba bekerül a kapualláncból leválasztott szám. Ha az új helyiség száma 92, akkor a játékos nem jut új helyiségbe, hanem verembe zuhan és meghal. A 286-os sor intézi el az esetet: kiír egy üzenetet, majd a játékos halálát kezelő szubrutin következik. Egyébként a játékos X,Y koordinátáiba bekerül a kapualláncból leválasztott érték, és a Vezérlőciklusba egy olyan korai ponton lépünk be, ami megrajzolja az akciómezőt.

A Take kezelő

A V villentyű lenyomása a 210-es és 220-as sorok között található Take kezelőt hívja be. Take a következő döntéseket hozza:

- Túl sok tárgy van a kalandozónál?
- Van valami mozdítható a közelben?
- A gyümölcsöt akarja fölvenni?
- Az italt akarja felvenni?
- Kincset akar felvenni?

A 212-es sor úgy kezdődik, hogy a K változóval leszámlálja a leltári vektort, és szabad helyet keres az új tárgynak. Ha nem talál, akkor kiírja: „*TÚL SOKAT AKAR VINNI*", és ezzel a Take befejeződik.

Ezután a 212-es sor végigpásztázza a közvetlen környezetet a mozdítható tárgyak után kutatva. A keresés a játékos fölött, tőle balra kezdődik. Az I és J változókkal futó, egymásba skatulyázott ciklusok megvizsgálják a játékost körülvevő háromszor hármas négyzetet. Ha a pásztázás olyan objektumra bukkan, ami nem lény, fal vagy tűz, és nem is a játékos, azt föl lehet venni. Eredménytelen keresés hatására megjelenik a „NINCS ITT SEMMI FÖLVEHETŐ!" üzenet, és a Vezérlő következik.

A 214-es sor törli a felvett tárgy helyét, majd ellenőriz néhány különleges esetet. Ha a kérdéses tárgy az ital, akkor a játékos teljes 10000 pontos ereje visszatér, megjelenik a megfelelő üzenet, és a gyümölcs új helyiségbe kerül. Ha a felvett tárgy a gyümölcs, a játékos ereje egy véletlen mennyiséggel növekszik, megjelenik a „FINOM VOLT!" üzenet, és a Take befejeződik.

A 216-os sor folytatja a különleges esetek vizsgálatát. Amikor a tárgy kincsnek bizonyul, meg kell álapítani, hogy a 32 kincs közül melyikről van szó. Egy ciklus megvizsgálja, mely kincsek vannak a helyiségben; feltételezzük, hogy azt vette föl a játékos, amelyiket a ciklus elsőnek megtalál.

Akár kincs, akár sem, a tárgy bekerül a hátizsákba: vagyis a helyzetét jelző szám 91 lesz, és beillesztjük a C(K) tömbelembe — K ugyanis az első szabad tömbelemet mutatja. Egy záróüzenet tudatja a művelet befejeződését, és a program a Vezérlőciklusba tér vissza.

A Drop kezelő

Az 1 és 8 közé eső számjegyek valamelyikét megnyomva sor kerül a Drop kezelő meghívására. Drop a 230-as és a 236-os sor között található.

A 230-as sor — Take-hez hasonlóan — végigpásztázza a játékost körülvevő háromszor hármas négyzetet. Ezúttal azonban Drop csak egy szabad helyet keres a lerakandó tárgynak. Feltéve, hogy nem talál szóközt (32-es kódú karakter), úgy tesz, mintha a játékos teljesen be lenne kerítve, és nem engedélyezi a tárgy lerakását, amit a „NEN LEHET SEHOVA LETENNI!" üzenet tudat.

Különben a 232-es sor előveszi a lenyomott billentyű ASCII kódját — ez még mindig a D változóban van —, kivon belőle 48-at, ezzel az 1 és 8 közötti számtartományba transzformálja. A leltári tárgyak vektorából azt a tárgyat rakjuk le, amelyikre az a szám mutat. A tárgy helyzetét jelző szám a mostani helyiség száma lesz. A 232-es sor utolsó részének az a feladata, hogy a kincseket — 16-nál nagyobb sorszámú mozdítható tárgyakat — a megfelelő jellel rajzoljuk meg.

A 234-es sor POKE utasítással elhelyezi a lerakott tárgyat a képernyő szabad pontjára. Ezután a 236-os sor tömöríti C(*n*) megmaradt elemeit, hogy ne legyen köztük hézag. Ezzel Drop-nak vége, és visszatérünk a Vezérlőhurokba.

A Quit kezelő

A K billentyű lenyomása meghívja a 240-es és 246-os sor között található Quit kezelőt. Ez a következőket teszi:

- értékeli a felkutatott kincseket,
- értékeli a megölt lényeket,
- értékeli, hányszor halt meg a játékos,
- kiírja az érvényes pontszámot,
- módot ad rá, hogy a játékot befejezzük.

A 240-es sor nullát tölt a J változóba; J számlálja a pontokat. A kincseket úgy értékeljük, hogy egy ciklussal megvizsgáljuk az objektumok helyzetét tároló vektort és kikeressük a bázison — az 1-es helyiségben — tárolt kincseket. Minden egyes biztonságba helyezett kincsért a kincs számával arányos pontot adunk J-hez. A kincsek pontértéke 17 és 48 között van.

Ezután a 242-es sor értékeli a megölt lényeket. A vizsgálóciklus pontot ad a nullás helyiségben található lényekért — ezeknél L(*n*) értéke nulla, ugyanis mind kimúltak. A kapott pontok száma 9-től 16-ig terjed, a legyőzött lény méretétől függően.

Végül az összegből levonjuk, hányszor halt meg a játékos. L(100)-ban találjuk, hányszor halt meg a játékos, és minden egyes haláláért 30 pontot vonunk le az összegből.

A 224-es sor PRINT utasítással — magyarázó szöveg kíséretében — kiírja a teljes érvényes pontszámot. A szöveg elfoglalja az állapotmező nagy részét. Ezt követően a 246-os sor választ vár a játékostól. Ha a „BEFEJEZI?" kérdésre „N" választ adunk, akkor egy ciklus törli a teljes állapotmezőt, és egy olyan ponton lépünk be a Vezérlőciklusba, ahol a Status szubrutin újra megrajzolja az állapotmezőt. „I" válaszra a fénypont a képernyő bal felső sarkába kerül, majd az END utasítással befejezzük a BASIC programot. Bármilyen más válasz esetén a program a 246-os sorban várakozik, megakadályozva, hogy hibás válasz miatt fejeződjön be a játék.

A Shoot kezelő

Az „L" billentyű hatására a Shoot kezelő végrehajtása következik. A kezelő a 250-es sorban kezdődik. Shoot-nak a következő eshetőségekre kell felkészülnie:

- Van íja a játékosnak?
- Van nyila a játékosnak?
- Van a helyiségben lény?
- Nem túl kemény ellenfél a lény a nyílnak?
- Célt téveszt a lövés?

A 250-es sor vizsgálja meg az első három esetet. Az L(*n*) helyzetjelző vektorból Shoot megállapíthatja, hogy az íj a játékosnál van-e: ebben az esetben a helyzetjelző szám 91. Ha nincs nála, akkor megjelenik a „NINCS ÍJA" üzenet, és a kezelő befejeződik. Hasonlóan, a nyíl hiányára a „NINCS NYILA" üzenet figyelmeztet. L(104) vizsgálatával Shoot eldöntheti, van-e lény a helyiségben. Ha nincs, megjelenik a „SZISSZ!!" üzenet, és a nyílvesszőt vaktában kilőjük.

A 256-os sor elhelyezi a kilőtt nyilat az akciómezőben. Randxy meghívása megad egy lehetséges helyet. A hozzá tartozó L(*n*) elembe a helyiség száma kerül, és a nyíl szimbólumát POKE-kal beírjuk a megadott helyre. Ezután meghívjuk a SUBINV szubrutint. Ennek az a feladata, hogy egy bizonyos tárgyat eltávolítson a hátizsákból. A szubrutin a 910-es sorban található.

SUBINV a lerakandó tárgy sorszámát az A változóban várja. Ciklusban végigfuttatja a C(*n*) leltári vektor elemeit, míg a keresett tárgyat megtalálja. Az elemet ezután úgy törli, hogy a rá következőket eggyel előbbre másolja. C(8)-ba mindig nulla kerül, mivel a sohasem használt C(9) tartalmát töltjük bele.

Miután meghívtuk Subinv-et, hogy a nyilat vegye ki a hátizsákból, a lény visszavonulását vezérlő változót úgy állítjuk be, hogy a lény hátráljon. Most

ugyan nincs lény, de ezeknek a változóknak a megváltoztatása biztosan nem árt ilyen esetekben. Hamarosan ugyanezt a 256-os sort felhasználjuk abban az esetben, mikor a lövés távozásra bírja a bestiát.

Feltételezve, hogy a helyiségben van lény, a 252-es sor megvizsgálja, hogy milyen erős. A nyílvessző levágódik az 5000-nél erősebb lény bőréről. Ebben az esetben megjelenik a „LEVÁGÓDIK RÓLA!" üzenet, és a 256-os sor a helyiségben találomra lerakja valahol a használhatatlan nyílvesszőt, egyébként a nyílvessző célba találhat.

Úgy döntjük el, hogy a nyílvessző célba talált-e, hogy kiszámítjuk a kalandozó és a lény távolságát — négyzetgyököt vonunk X és Y négyzeteinek összegéből. A távolság 1 és 55 közé esik. A távolságot 81-ből kivonjuk, ez lesz az alapja a százalékos próbának. Egy véletlenszerűen kiválasztott százalékos értéket összehasonlítunk az előzőleg kapott számmal úgy, hogy annál nagyobb a találat valószínűsége, minél közelebb a cél. A találat valószínűsége legföljebb 80 százalék lehet.

Ha a lövés célt téveszt, akkor a következő üzenet jelenik meg: „AZ ÖRDÖGBE! NEM TALÁLT!", és a 256-os sor ismét találomra lerakja valahol a nyílvesszőt a helyiségben.

Különben a 254-es sor a helyiség számát írja be a nyílvessző helyét jelölő változóba, és a nyíl szimbóluma a képernyőn felülírja a megölt lény szimbólumát. Meghívjuk Subinv-et, hogy a nyílvesszőt kitörölje a játékosnál levő tárgyak vektorából. A lény helyét jelölő változóba nulla kerül, hasonlóan L(104)-hez. Ez utóbbi azt jelzi, hogy a helyiségben immár nincs lény. Megjelenik a felirat: „ELTALÁLTA! GYŐZELEM!", és a kezelő befejeződik.

A Fight kezelő

A „H" bilentyű lenyomására a Fight kezelő lép működésbe. A kezelő a 290-es és a 298-as sor között van. Fight a következő eseteket mérlegeli:

- Esetleg a játékosnak nincs is kardja.
- Esetleg nincs is lény, amivel meg lehet küzdeni.
- Nincs túl távol a lény ahhoz, hogy kárt tegyünk benne?

A 290-es sor megvizsgálja a tárgyak helyét tároló vektort, hogy megállapítsa: a játékosnál van-e a kard. Ha nincs, akkor az üzenetablakban megjelenik, hogy „NINCS IS KARDJA". Ha L(104)-ből az derül ki, hogy a helyiségben nincs is lény, akkor a következő csípős megjegyzés jelenik meg: „ÁRNYÉKOKKAL AKAR HARCOLNI?".

Az X és Y koordináták különbségének abszolút értéke alapján megállapítható, hogy milyen közel kell lennie a lénynek ahhoz, hogy a karddal lesújthassunk rá. A 292-es sorban elvégzett vizsgálat után, amikor a lény és a játékos nem érintkeznek közvetlenül, a „FUJ! ELHIBÁZTA!" üzenet jelenik meg.

Ezután a 298-as sor 5 százalékkal csökkenti a játékos erejét. Ha ezzel a játékos ereje elfogyott, akkor meghal, és a halált kezelő 128-as szubrutin következik.

Tegyük fel, hogy a kard célba talál, akkor a 294-es sor a játékos erejének 20 százalékát levonja a lény erejéből. Lehet, hogy ez nem meríti ki a lény erejét, tehát a „JÓL SIKERÜLT CSAPÁS" üzenet jelenik meg. Ezenkívül 50–50 százalék esélye van annak, hogy a támadást követően a lény megfutamodik — hátrálni kezd. Ha a csapás kimeríti a lény erejét, a 296-os sor kiírja, hogy „MEGÖLTE!". A lény a 0-s helyiségbe távozik, a helyiség lénymentesnek nyilváníttatik és a lény szimbóluma törlődik. Megölte-e a csapás a lényt, vagy sem, a 298-as sor mindenképp csökkenti a lény erejét.

13. FEJEZET

A Szörnyek az útvesztőben listája

Hát ennyi az egész! Már csak az hiányzik, hogy beírjuk a gépbe, és a *Szörnyek az útvesztőben* működni kezd. Ebből a célból következzék tehát a teljes lista!

Ha már az Olvasó futtatni fogja az „Útvesztőt", esetleg javítani valót is talál benne. lehet, hogy a pontozásnál használt értékek nem tetszenek. Esetleg további parancsokat vagy másfajta lényeket szeretne. Szinte bizonyosra veszem, hogy változtatni szeretne az egyes helyiségek formáján. Pontosan úgy, mint a szöveg típusú játékban: a határ a csillagos ég — de legalább is a tár mérete.

```
                     INICIALIZALO RÉSZ
2 CLS:PRINTCHR$(23):PRINT@532,"Üdvözöljük a
":PRINT@587,"SZÖRNYEK AZ UTVESZTÖBEN":PRINT
@664,"játékban";:CLEAR256:DEFINTA-Z:DIML(10
7),C(9):DEFSTRR:DIMR(90):GOSUB1000
4 FORN=17TO48:L(N)=RND(87)+3:NEXTN
6 X$=STRING$(90," "):P=PEEK(VARPTR(X$)+1)+P
EEK(VARPTR(X$)+2)*256-1:FORN=49TO96
8 M=RND(87)+3:IFPEEK(P+M)<>32THEN8ELSEPOKEP
+M,45:L(N)=M:NEXTN:X$=""
10 TA$="FáklyaABCDEFPortálABCDEFABCDEFABCDE
FABCDEFPajzs Ij      ABCDEFItal   ABCDEFNyíl
Alma   Kard   Bomba  ":TB$="Pók        Kigyo    Rá
k      Skorpio ÓriásméhAmöba    Törpe    Sárká
ny  "
14 L(3)=RND(87)+3:L(8)=RND(87)+3:L(9)=RND(8
7)+3:L(11)=RND(87)+3:L(13)=RND(87)+3:L(16)=
RND(87)+3
16 L(1)=1:L(15)=1:L(0)=1:L(97)=28:L(98)=0:L
(99)=10000:CLS
```

13-1. ábra. A SZÖRNYEK AZ ÚTVESZTŐBEN inicializáló része

VEZÉRLŐ RÉSZ

```
100 GOSUB920:GOSUB500:IFL(104)<>0THENL(103)
=(L(104)-49)*150+5000:PRINTé56,"Vigyázat";:
PRINTé120,MID$(TB$,FIX((L(104)-49)/6)*8+1,8
);:L(105)=1
102 GOSUB900
104 PRINTé1016,L(99);:GOSUB200:IFD=0THEN110
ELSEIFD>7ANDD<11ORD=91THEN260ELSEGOSUB920:I
FD<57ANDD>48THEN230ELSEIFD<70ORD>90THEN110
106 OND-64GOTO110,110,110,110,110,110,110,2
90,110,110,240,250,110,110,110,110,110,110,
110,110,110,210,110,110,110,110
110 IFL(104)=0THEN126ELSEX=L(101)+SGN(L(97)
-L(101))*L(105):Y=L(102)+SGN(L(98)-L(102))*
L(105):D=PEEK(X+64*Y+15360):IFD<>64THEN116
112 GOSUB920:PRINTé56,MID$(TB$,FIX((L(104)-
49)/6)*8+1,8);:PRINTé120,"Támad!!!";:POKEL(
97)+64*L(98)+15360,191:POKEL(97)+64*L(98)+1
5360,64:L(105)=-1:L(106)=RND(20):L(99)=L(99
)-L(103)/8:IFL(8)=91THENL(99)=L(99)+500
114 IFL(99)>0THEN122ELSE128
116 IFD=48THENFORI=1TO20:POKEX+64*Y+15360,R
ND(64)+128:NEXT:POKEX+64*Y+15360,32:L(16)=R
ND(87)+3:GOTO124
117 IFD=46THENPOKEX+64*Y+15360,32:L(99)=L(9
9)+RND(4000)+2000:GOTO120:ELSEIFD=43THENPOK
EX+64*Y+15360,32:L(99)=10000:L(11)=RND(87)+
3:GOTO120
118 IFD=32THENPOKEL(101)+64*L(102)+15360,32
:L(101)=X:L(102)=Y:POKEX+64*Y+15360,42:GOTO
120
119 IFL(105)=-1THENL(105)=1:ELSEFORI=1TO3:X
=L(101)+RND(3)-2:Y=L(102)+RND(3)-2:D=PEEK(X
+64*Y+15360):IFD<>32THENNEXT:ELSE118
120 IFL(105)=-1THENL(106)=L(106)-1:IFL(106)
<=0THENL(105)=1
122 IFL(107)>1THEN126ELSEL(103)=L(103)-1:IF
L(103)>0THEN126
124 POKEL(101)+64*L(102)+15360,32:PRINTé56,
"Végre!!!";:PRINTé120,"Kimult  ";:L(L(104))
```

```
=0:L(104)=0
126 L(107)=L(107)-1:IFL(107)>0THEN104ELSEL(
107)=10:L(99)=L(99) 1:IFL(99)>0THEN104
128 FORI=1TO8:L(C(I))=L(0):C(I)=0:NEXTI:L(1
00)=L(100)+1:CLS:PRINTÉ512,"Sajnos Ön kimul
t. Szerencsére fel tudjuk támasztani!
Ha felkészült, nyomja meg a <NEWLINE> bille
ntyűt!";
130 X$=INKEY$:IFX$=CHR$(13)THENCLS:GOTO16:E
LSE130
```

. ábra. A **SZÖRNYEK AZ ÚTVESZTŐBEN vezérlőciklusa**

```
BILLENTYU KEZELO

200 D=PEEK(14400):IF(DAND32)=32THEND=8:RETU
RN:ELSEIF(DAND64)=64THEND=9:RETURN:ELSEIF(D
AND8)=8THEND=91:RETURN:ELSEIF(DAND16)=16THE
ND=10:RETURN
204 D=PEEK(14352):IFD=0THEN206ELSEFORI=1TO7
:IF(DAND(2£I))<>0THEND=48+I:RETURNELSENEXTI

206 D=PEEK(14338):IFD=0THEN207ELSEIF(DAND1)
<>0THEND=72:RETURN:ELSEIF(DAND8)<>0THEND=75
:RETURN:ELSEIF(DAND16)<>0THEND=76:RETURN
207 IFPEEK(14368)AND1<>0THEND=56:RETURN
208 IFPEEK(14340)AND64<>0THEND=86:RETURN
209 D=0:RETURN
```

```
KEZELÖK

TAKE

T billentyüt lenyomni

210 FORK=1TO8:IFC(K)<>0THENNEXTK:PRINT£56,"
Ez már  ";:PRINT£120,"tul sok ";:GOTO110
212 N=L(97)+64*L(98)+15295:FORI=NTON+128STE
P64:FORJ=0TO2:A=PEEK(I+J)-32:IFA<>0ANDA<17A
NDA<>10ANDA<>6THEN214ELSENEXTJ,I:PRINT£56,"
Mit akar";:PRINT£120,"elvenni?";:GOTO110
214 POKEI+J,32:IFA=11THENL(99)=10000:PRINT£
56,"Eröm a ";:PRINT£120,"régi !";:L(11)=RND
(87)+3:GOTO110:ELSEIFA=14THENL(99)=L(99)+RN
D(4000)+2000:PRINT£56,"Csuda.jo";:PRINT£120
,"volt !";:GOTO110
216 IFA=4THENFORA=17TO48:IFL(0)<>L(A)THENNE
XTA
218 L(A)=91:C(K)=A
220 PRINT£56,"Rendben!";:PRINT£120,"
 ";:GOTO102
```

13-3. ábra. A Take kezelő

```
DROP

Számjegy billentyüt lenyomni


230 N=L(97)+64*L(98)+15295:FORI=NTON+128STE
P64:FORJ=0TO2:IFPEEK(I+J)<>32THENNEXTJ,I:PR
INT£56,"Nem fér";:PRINT£120,"sehova!";:GOTO
110
232 K=C(D-48):L(K)=L(0):IFK>16THENK=4
234 POKEI+J,K+32
236 FORI=D-48TO8:C(I)=C(I+1):NEXTI:GOTO102
```

13-4. ábra. A Drop kezelő

```
                QUIT

  Q billentyüt lenyomni


240 J=0:FORI=17TO48:IFL(I)<>1THENNEXT:ELSEJ
=J+I:NEXT
242 FORI=49TO96:IFL(I)<>0THENNEXT:ELSEJ=J+F
IX((I-1)/6)+1:NEXT
244 J=J-L(100)*30:PRINTé184,"Ha most ";:PRI
NTé248,"fejezné ";:PRINTé312,"be a    ";:PR
INTé376,"játékot ";:PRINTé440,"          ";:P
RINTé440,J;:PRINTé504,"pontja  ";:PRINTé568
,"lenne.  ";:PRINTé632,"Fejezzük";:PRINTé69
6,"be?      ";:PRINTé760,"I vagy N";
246 X$=INKEY$:IFX$="N"THENFORI=0TO14:PRINTé
56+I*64,"            ";:NEXT:GOTO102ELSEIFX$="I
"THENPRINTé0,"";:END:ELSE246
```

13-5. ábra. A Quit kezelő

```
                SHOOT

  S billentyüt megnyomni


250 IFL(9)<>91THENPRINTé56,"Nincs is";:PRIN
Té120,"ijja  !";:GOTO110:ELSEIFL(13)<>91THE
NPRINTé56,"Nincs is";:PRINTé120,"nyila   ";
:GOTO110:ELSEIFL(104)=0THENPRINTé56,"ZZINGO
!!";:PRINTé120,"'           ";:GOTO256
252 IFL(103)>5000THENPRINTé56,"Lepattan";:P
RINTé120,"rola!   ";:GOTO256:ELSEN=SQR((L(10
1)-L(97))é2+(L(102)-L(98))é2):N=81-N:IFRND(
100)>NTHENPRINTé56,"Hibázott";:PRINTé120,"a
jjaj! ";:GOTO256
254 L(13)=L(0):POKEL(101)+64*L(102)+15360,4
5:A=13:GOSUB910:L(L(104))=0:L(104)=0:PRINTé
56,"Találat!";:PRINTé120,"Éljen! ";:GOTO102

256 GOSUB940:L(13)=L(0):POKEX+64*Y+15360,45
:A=13:GOSUB910:L(105)=-1:L(106)=RND(20):GOT
O102
```

13-6. ábra. A Shoot kezelő

```
                MOVE

 Nyil billentyüt megnyomni


260 X=0:Y=0:IFD=91THEND=11
262 OND-7GOTO264,266,268,270
264 X=-1:GOTO272
266 X=1:GOTO272
268 Y=1:GOTO272
270 Y=-1:GOTO272
272 X=X+L(97):Y=Y+L(98):Q=PEEK(X+64*Y+15360
)
274 IFQ=32THENPOKEL(97)+64*L(98)+15360,32:L
(97)=X:L(98)=Y:POKEX+64*Y+15360,64:L(99)=L(
99)-5:IFL(99)>0THEN110ELSE128
276 IFQ=48THENPRINT@56,"BUMM!";:PRINT@120,"
Szamár!";:FORI=1TO40:POKEX+64*Y+15360,RND(6
4)+128:NEXTI:L(16)=RND(87)+3:GOTO128
278 IFQ=35THENI=RND(87)+3:IFL(1)<>91ANDL(1)
<>ITHENGOSUB920:PRINT@56,"Semmi se";:PRINT@
120,"történik";:GOTO110:ELSEPRINT@X+64*Y,"P
UFF";:FORI=1TO20:NEXTI:L(0)=I:GOTO100
280 IFQ=38THENPOKEL(97)+64*L(98)+15360,32:L
(97)=X:L(98)=Y:POKEX+64*Y+15360,64:L(99)=L(
99)-RND(200)+100:GOSUB920:PRINT@56,"TÜÜZ!!"
;:PRINT@120,"ÉGET!!!";:IFL(99)>0THEN110ELSE
128
282 IFQ=128THENIFL(1)<>91THEN288ELSEFORI=1T
O91STEP10:IFX=VAL(MID$(CR$,I,2))ANDY=VAL(MI
D$(CR$,I+2,2))THENL(0)=VAL(MID$(CR$,I+4,2))
:IFL(0)=92THEN286ELSEL(97)=VAL(MID$(CR$,I+6
,2)):L(98)=VAL(MID$(CR$,I+8,2)):GOTO100:ELS
ENEXTI
284 GOTO110
286 PRINT@56,"Jaj egy!";:PRINT@120,"gödör";
:FORI=1TO20:NEXT:GOTO128
288 GOSUB920:PRINT@56,"Sötét    ";:PRINT@120
,"van benn";:GOTO110
```

13-7. ábra. A Move kezelő

```
               FIGHT

    F billentyüt megnyomni


290 IFL(15)<>91THENPRINTé56,"Nincs is";:PRI
NTé120,"kardja! ";:GOTO110:ELSEIFL(104)=0TH
ENPRINTé56,"Árnnyal ";:PRINTé120,"harcol? "
;:GOTO298
292 IFABS(L(97)-L(101))>1ORABS(L(98)-L(102)
)>1THENPRINTé56,"Hibázott";:PRINTé120,"panc
ser!";:GOTO298
294 L(103)=L(103)-L(99)/5:IFL(103)>0THENPRI
NTé56,"Telibe";:PRINTé120,"találta!";:IFRND
(2)=2THENL(105)=-1:L(106)=RND(20):GOTO298:E
LSE298
296 PRINTé56,"Megölte ";:PRINTé120,"Vége!
 ";:L(L(104))=0:L(104)=0:POKEL(101)+64*L(10
2)+15360,32
298 L(99)=L(99)-L(99)/20:IFL(99)>0THEN110EL
SE128
```

13-8. ábra. A Fight kezelő

SZUBRUTINOK

DSPLAY

```
500 L(104)=0:CR$=R(L(0)):PRINT@0,STRING$(56
,191);:PRINT@960,STRING$(56,191);:FORI=64TO
896STEP64:PRINT@I,CHR$(191);STRING$(54,32);
CHR$(191);:NEXTI:N=1
502 IFMID$(CR$,N,1)="-"THENN=N+1:GOTO504ELS
EX=VAL(MID$(CR$,N,2)):Y=VAL(MID$(CR$,N+2,2)
):POKEX+64*Y+15360,128:N=N+10:GOTO502
504 IFN>LEN(CR$)THEN520ELSEX=VAL(MID$(CR$,N
+1,2)):Y=VAL(MID$(CR$,N+3,2)):ONVAL(MID$(CR
$,N,1))GOTO506,508,508,508,510,512:GOTO520
506 PRINT@X+64*Y,STRING$(VAL(MID$(CR$,N+5,2
)),191);:N=N+7:GOTO504
508 S=X+64*Y+15360:D=VAL(MID$(CR$,N,1))+61:
FORI=1TOVAL(MID$(CR$,N+5,2)):POKES,191:S=S+
D:NEXTI:N=N+7:GOTO504
510 S=X+64*Y-65:PRINT@S,"&&&";:S=S+64:PRINT
@S,"&&&";:S=S+64:PRINT@S,"&&&";:N=N+5:GOTO5
04
512 POKEX+64*Y+15360,46:N=N+5:GOTO504
520 POKEL(97)+64*L(98)+15360,64:FORI=1TO16:
IFL(I)=L(0)THENGOSUB940:POKEN,I+32
522 NEXTI:FORI=17TO48:IFL(I)=L(0)THENGOSUB9
40:POKEN,36
524 NEXT:FORI=49TO96:IFL(I)=L(0)THENGOSUB94
0:POKEN,42:L(101)=X:L(102)=Y:L(104)=I:ELSEN
EXTI
526 RETURN
```

13-9. ábra. A Dsplay kezelő

STATUS

```
900 PRINT@248,"hely";L(0);:PRINT@952,"Ereje
:  ";:PRINT@1016,L(99);
902 FORI=1TO8:IFC(I)=0THENPRINT@312+I*64,"
       ";:RETURN:ELSEPRINT@312+I*64,CHR$(I+
48);" ";:IFC(I)>16THENPRINT"ki";C(I)-16;:NE
XT:RETURN:ELSEPRINTMID$(TA$,C(I)*6-5,6);:NE
XT:RETURN
```

13-10. ábra. A Status szubrutin

```
SUBINV

910 FORI=1TO8:IFC(I)<>ATHENNEXTI:RETURN:ELS
EFORJ=ITO8:C(J)=C(J+1):NEXTJ:RETURN
```

13-11. ábra. A Subinv szubrutin

```
CLRMES

920 PRINT@56,"             ";:PRINT@120,"
 ";:RETURN
```

13-12. ábra. A Clrmes szubrutin

```
RANDXY

940 X=RND(54):Y=RND(14):N=X+64*Y+15360:IFPE
EK(N)<>32THEN940ELSERETURN
```

13-13. ábra. A Randxy szubrutin

HELYISéGEK LANCAI

```
1000 R(1)="00060254085507030106270090271 4-4
010111254011111180418"
1001 R(2)="550801010619000423140006905540820
15062501-306011231203121130739"
1002 R(3)="000601540721150727015507080109 22
00092914-101102732704071280422"
1003 R(4)="231502190124001031140100114014-3
040108105084531909 04"
1004 R(5)="550802010900021254110012135403-1
010850351040950202"
1005 R(6)="250002201401151441012615153201-3
050609106062733107 04"
1006 R(7)="270003211428151633013915171401-1
040451"
1007 R(8)="000903540755101801055502190110-3
200105324031255310"
1008 R(9)="291503220140002015143000213414-2
360112436 0112"
1009 R(10)="31150424010006225406-3310408127
0909"
1010 R(11)="40150401011000224414-1080538246
0510"
1011 R(12)="55110501020012235402-1011020324
1005"
1012 R(13)="55030501120004235412-4210106233
0106"
1013 R(14)="41000601141315244501-1011230330
021:"
1014 R(15)="32000626140007245407-1010538"
1015 R(16)="33000728145508250108-3450108121
0928"
1016 R(17)="14000739144215251601-1010420401
0507"
1017 R(18)="00050854105501260113-3170110321
04113250110"
1018 R(19)="00100854025511260103-1040943120
0520"
1019 R(20)="15150940014300271714-4420109120
0931"
1020 R(21)="34150930015509270109-1080529108
```

13-14. ábra. A helyiségek láncai

```
10293360510"
1021 R(22)="550610010644151110010315285404-
22896084280609"
1022 R(23)="550212011255121301040008285408-
101053510110353360504"
1023 R(24)="450014131455071501070900285414-
25401102440110"
1024 R(25)="000816540816001742145000290110-
34901051030545"
1025 R(26)="00131854010003195421550029010[illegible]-
31[illegible]"
1026 R(27)="1715204301000092154095215290105-
3180411"
1027 R(28)="5504220314550823010855142409014
615300301-45002052540905103064460510"
1028 R(29)="0010255001000726540800052752140
415305001-40106071081238146040961804"
1029 R(30)="030028461450002904142715312901-
32801101060542"
1030 R(31)="000632540855073301062900302714-
401011125401111180418"
1031 R(32)="5508310106190034231400093554082
015362501-306011231203121130739"
1032 R(33)="0006315407211537270155073801092
200392914-101102732704071280422"
1033 R(34)="231532190124004031140100414014-
304010810508453190904"
1034 R(35)="550832010900024254110012435403-
1010850351040950202"
1035 R(36)="250032201401154441012615453201-
305060910606273310704"
1036 R(37)="270033211428154633013915471401-
1040451"
1037 R(38)="000933540755104801055502490110-
3200105324031255310"
1038 R(39)="291533220140005015143000513414-
23601124360112"
1039 R(40)="31153424010006525406-3310408127
0909"
1040 R(41)="40153401011000524414-1080538246
```

```
0510"
1041 R(42)="55113501020012535402-101102032
41005"
1042 R(43)="55033501120006453541 2-4210106233
0106"
1043 R(44)="410036011413155445 01-1011230330
0212"
1044 R(45)="32003626140007545407-1010538"
1045 R(46)="33003728145508550108-3450108121
0928"
1046 R(47)="14003739144215551601-1010420401
0507"
1047 R(48)="00053854105501560113-3170110321
04113250110"
1048 R(49)="00103854025511560103-1040943120
0520"
1049 R(50)="15153940014300571714-4420109120
0931"
1050 R(51)="34153930015509571019-1080529108
10293360510"
1051 R(52)="5506400106441541100103155854O4-
22806084280609"
1052 R(53)="55024201125512430104000858540 8-
101053510110353360504"
1053 R(54)="45004413145507450107090058541 4-
25401102440110"
1054 R(55)="00084654081600474214500059011 0-
34901051030545"
1055 R(56)="00134854010003495411550859010 7-
315010731509063450110"
1056 R(57)="17155043010009515409521559010 5-
3180411"
1057 R(58)="55045203145508530108551454090 14
615600301-4500205254090510306446 0510"
1058 R(59)="00105550010007565408000557521 40
415605001-4010607108123814604096 1804"
1059 R(60)="03005846145000590414271561290 1-
32801101060542"
1060 R(61)="00066254085507630106290060271 4-
40101112540111180418"
```

```
1061 R(62)="5508610106190064231400096554082
015662501-306011231203121130739"
1062 R(63)="0006615407211567270155076801092
200692914-101102732704071280422"
1063 R(64)="231562190124007031140100714014-
304010810508453190904"
1064 R(65)="550862010900027254110012735403-
1010850351040950202"
1065 R(66)="250062201401157441012615753201-
305060910606273310704"
1066 R(67)="270063211428157633013915771401-
1040451"
1067 R(68)="000963540755107801055502790110-
3200105324031255310"
1068 R(69)="291563220140008015143000813414-
23601124360112"
1069 R(70)="31156424010006825406-3310408127
0909"
1070 R(71)="40156401011000824414-1080538246
0510"
1071 R(72)="55116501020012835402-1011020324
1005"
1072 R(73)="55036501120004835412-4210106233
0106"
1073 R(74)="41006601141315844501-1011230330
0212"
1074 R(75)="32006626140007845407-1010538"
1075 R(76)="33006728145508850108-3450108121
0928"
1076 R(77)="14006739144215851601-1010420401
0507"
1077 R(78)="00056854105501860113-3170110321
04113250110"
1078 R(79)="00106854025511860103-1040943120
0520"
1079 R(80)="15156940014300871714-4420109120
0931"
1080 R(81)="34156930015509870109-1080529108
10293360510"
1081 R(82)="550670010644157110010315885404-
```

```
22806084280609"
1082 R(83)="55027201125512730104000888540 8-
10105351011035336050 4"
1083 R(84)="45007413145507750107690088541 4-
25401102440110"
1084 R(85)="00087654081600774214500089011 0-
34901051030545"
1085 R(86)="00137854010003795411530889010 7-
3150107315090634501 10"
1086 R(87)="17158043010009815409521589010 5-
3180411"
1087 R(88)="550482031455088361085514840 9014
615900301-4500205254090510306446 0510"
1088 R(89)="001085500100078654080005875214 0
415905001-4010607108123814604096 1804"
1089 R(90)="03008846145000890414271501290 1-
32801101060542"
1090 RETURN
```

›sszefoglalás

[ogyan is lehet egy teljes könyvet összefoglalni? Véleményem szerint az a ;gjobb módszer, hogy átismételjük a legfontosabb alapelveket: így egyes lvasók számára tisztázzuk a dolgokat, megkockáztatva, hogy másokat unta-ınk vele.

Akik tovább olvasnak, azok összpontosítsák figyelmüket az első alap-lvre: a program szerkezetére! A legegyszerűbb programokat kivéve elkerül-etetlen, hogy a programot többszöri próbálgatás, majd javítás árán készítsük l: egy strukturálatlan programot pedig nagyon nehéz javítani. A nehézségek gy része egyenesen a struktúra hiányára vezethető vissza. Ilyen pl. az, amikor :ándékunk ellenére többször használunk egy változót — legjobb, ha azt rögtön lfelejtjük. Érdemes rászánnunk egy órát a folyamatábra vagy egy táblázat lkészítésére, mert ez a későbbiekben bőven megtérül: tízórányi zűrzavart ıkaríthatunk meg a billentyűk mellett.

A második alapelv még fontosabb: vegyük figyelembe az összes lehető-;get! Egy olyan sok beolvasást tartalmazó programnál is, amilyen a *Kardhalak : kincsek*, biztosan akad valaki, aki nem várt módon fogalmaz meg egy paran-ıot, vagy olyan dolgot próbál megtenni, amire nem számítottunk, és a hatás ;etleg katasztrofális a programra nézve. próbáljunk minél többet gondolkodni :on, hogy „mi történik abban az esetben, ha...”, és alaposan próbáljuk ki a :ogramot, mielőtt átadnánk egy kalandozójelöltnek!

Az utolsó alapelv megvalósítása jó sok időt igényel: optimalizáljuk a kódot! eressük meg az utasítások legegyszerűbb formáját, és gyorsítsuk a program ;grehajtását!

Használjunk ON-GOTO elágazóutasításokat és szubrutinhívó GOSUB :asításokat! Olvassunk el mindent, amihez hozzáférünk arról, hogyan is űködik a BASIC, és találjuk ki, hogyan lehet PEEK-kel és POKE-kal be-ılyásolni a működését! Amikor úgy véljük, hogy a „kalandprogram készíté-:n” a csúcsra értünk, meg fogunk lepődni, hogy még mennyi kipróbálandó ıgás maradt. Valószínű, hogy az Olvasó maga talál rá néhányra elsőnek.

Végtére is, ha az Olvasó készen áll arra, hogy egy szekercével a kezében lvegye a harcot a sárkányokkal, talán ahhoz is elég vakmerő, hogy progra-ozni kezdjen!

Tárgymutató

, GY

:

:

L

M

O, Ö

P

Q

- Yoco -

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Fischer Herbert igazgató
A Műszaki Könyvkiadó fényszedése

Alföldi Nyomda, Debrecen
86.2264.66-13-2

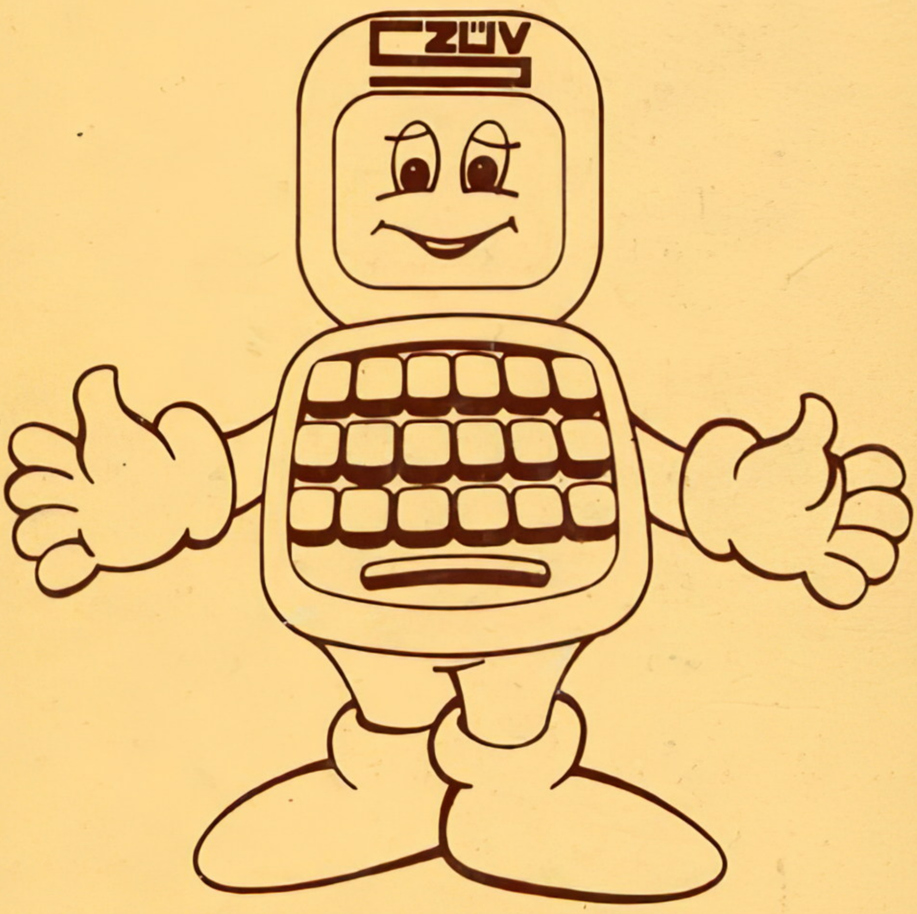
Műszaki vezető: Kőrizs Károly
Műszaki szerkesztő: Bagi Miklós
A borítót és a kötést tervezte: Kováts Tibor
A könyv ábráit rajzolta: Hack Emil
A könyv formátuma: A/5
Ívterjedelme: 14,5 (A/5)
Ábrák száma: 104
Papír minősége: 80 g ofszet
Betűcsalád és -méret: New Times, bg/gm
Azonossági szám: 61 349
MŰ: 3838-i-8688
A kézirat lezárva: 1985. március
Készült az MSZ 5601 és 5602 szerint

Ára: 63 Ft

SZÜV

ÜGYFÉLSZOLGÁLATI IRODA

1067 Budapest
Lenin krt. 57-59,
Tel: 224-838
Telex:
22-7610 suvlk h



COMPUTER-M

KOMOLYABB KALANDOZÓKNAK AJÁNLJUK:

- A számítástechnikai adathordozók forgalmazása
- Szoftver-bemutatás és -eladás
- Helyszíni géphasználat, szaktanácsadás